

Universität
Rostock



Traditio et Innovatio

SIMULATION AND DIGITAL GAME-BASED LEARNING IN
SOFTWARE ENGINEERING EDUCATION

An Integrated Approach to Learn Software Engineering Methods

Dissertation
zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von:
Jöran Pieper

Rostock, 2. November 2016

REVIEWERS OF DISSERTATION:

Prof. Dr. Peter Forbrig
Fakultät für Informatik und Elektrotechnik, Universität Rostock

Prof. Dr. Michael Goedicke
paluno, The Ruhr Institute for Software Technology
Universität Duisburg-Essen

Prof. Dr. Oliver A. Lüth
Fakultät für Elektrotechnik und Informatik, Hochschule Stralsund

DATE OF SUBMISSION:

2 November 2016

DATE OF DEFENSE:

8 March 2017

Jöran Pieper: *Simulation and Digital Game-Based Learning in Software Engineering Education*, An Integrated Approach to Learn Software Engineering Methods, © 2. November 2016

*Der Schüler soll nicht Gedanken, sondern denken lernen;
man soll ihn nicht tragen, sondern leiten, wenn man will,
dass er in Zukunft von sich selbst zu gehen geschickt sein soll.*

— Immanuel Kant (1724–1804), deutscher Philosoph

A journey of a thousand miles begins with a single step.

— Laozi (604BC–531BC), Chinese philosopher

ABSTRACT

Software Engineering (SE) methods support a structured, disciplined and goal-oriented approach to SE endeavors. The industry, as well as curriculum guidelines, demand strong competencies in this field, which is, unfortunately, less intuitive accessible. Observations show that the holistic perspective on SE endeavors gets regularly lost by students in course projects. It is not easy for them to orientate inside of a given SE method and to utilize it in a systematic goal-oriented way. The *Integrated Approach* proposed in this thesis, introduces students to SE methods. It was designed explicitly to facilitate project-based learning approaches by lowering the cognitive load students face at conducting their project work. It aims at providing more opportunities for reflection and social interaction to promote learning and to let students experience the orientation and guidance provided by the SE concepts to enable the development of SE attitudes, where the provided practices and methods are not just perceived as further cognitive load but as actually supporting at keeping a holistic perspective on the SE endeavor. To face the high number of existing SE methods and to increase the transferability of competencies gained, the generalist *SEMAT Essence Kernel* approach, considering all essential dimensions of SE endeavors in a practice and process independent way, gets utilized. To address these objectives, two learning games were developed. The *Essence Kernel Puzzler* facilitates learning the common vocab of the Essence kernel and exploring relationships between its elements. The *Simulation Game*, based on a new declarative modeling approach, is a tightly linked, and reflective learning game integrating the essential concepts of the Essence kernel, including the dynamic steering of an SE endeavor, deep into the gameplay and enabling students to experience those concepts in a hazard-free challenging environment. It provides collaboration to foster discussion and reflection as well as motivating competition. Also, the *Essence Navigator* was developed to be both, integrated deeply into the gameplay and providing the same orientation and guidance in real project work. The introduced approach, deeply grounded in learning theories, integrates these tools and games into a number of phases, each addressing its respective learning objectives. The evaluation of a case study conducted with students indicates the utility of the developed approach as well as the usefulness of the developed learning games and tools.

ZUSAMMENFASSUNG

Software Engineering (SE)-Methoden unterstützen ein strukturiertes, diszipliniertes und zielgerichtetes Vorgehen bei der Bearbeitung von SE-Aufgaben. Sowohl Industrie als auch Empfehlungen für Bachelor- und Masterprogramme fordern solide Kompetenzen auf diesem Gebiet, welches jedoch schwer intuitiv zugänglich ist. Beobachtungen zeigen, dass der holistische Blick auf eine SE-Aufgabe in einem Kursprojekt schnell verloren geht, dass Studierende regelmäßig Schwierigkeiten haben, sich in einer gegebenen SE-Methode zu orientieren und diese systematisch und zielgerichtet einzusetzen. Der in dieser Dissertation vorgestellte *Integrierte Ansatz* führt Studierende in SE-Methoden ein. Er wurde explizit entworfen, um projekt-basiertes Lernen zu unterstützen, indem die kognitive Belastung Studierender bei der Bearbeitung ihres Kursprojekts gesenkt wird. Er zielt darauf ab, mehr lernfördernde Anlässe und Gelegenheiten für Reflexion sowie soziale Interaktion zu schaffen und Studierenden Lernerfahrungen zu ermöglichen, in welchen SE-Methoden nicht als zusätzlicher Ballast, sondern als echte Unterstützung bei der ganzheitlichen Betrachtung und Orientierung in einem komplexen Umfeld empfunden werden. Um der hohen Anzahl von SE-Methoden zu begegnen und den möglichen Transfer erworbener Kompetenzen zu erhöhen, wird hierbei der generalistische Ansatz des *SEMAT Essence Kernel* genutzt, welcher alle essentiellen Dimensionen einer SE-Aufgabe prozess- und praktik-unabhängig berücksichtigt. Um o.g. Ziele zu erreichen, wurden zwei digitale Lernspiele entwickelt. Der *Essence Kernel Puzzler* unterstützt beim Erlernen des Vokabulars des Essence Kernels sowie bei der Erkundung der Beziehungen zwischen dessen Elementen. Das *Simulation Game* basiert auf einem neuen deklarativen Ansatz zur Simulationsmodellierung. Es integriert die Konzepte des Essence Kernels, einschließlich der dynamischen Steuerung einer SE-Aufgabe, tief in das Spielerlebnis und ermöglicht es Spielern auf herausfordernde Weise, die Anwendung der Konzepte in einer gefahrlosen Umgebung zu erleben. Das Lernspiel bietet Zusammenarbeit zur Förderung von Diskussion und Reflexion innerhalb von Spielerteams sowie motivierenden Wettbewerb zwischen diesen Teams. Darüber hinaus wurde mit dem *Essence Navigator* ein Werkzeug entwickelt, welches tief in das Spielerlebnis eingebettet ist und dieselbe Orientierung und Unterstützung in einem realen Projektumfeld bietet. Der tief in Lerntheorien verwurzelte vorgestellte Ansatz integriert diese Werkzeuge und Spiele in einzelne Phasen, die jeweils der Erreichung spezifischer Lernziele dienen. Die Evaluation des Testeinsatzes im Rahmen einer Fallstudie mit Studierenden indiziert die Nützlichkeit des entwickelten Ansatzes sowie der entwickelten Lernspiele und Werkzeuge.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

- [1] Jöran Pieper. "Alles nur Spielerei? Neue Ansätze für digitales spielbasiertes Lernen von Softwareprozessen." In: *Tagungsband des 13. Workshops "Software Engineering im Unterricht der Hochschulen" 2013*. Ed. by Andreas Spillner and Horst Lichter. Vol. Vol-956. Aachen: CEUR Workshop Proceedings, Feb. 2013, pp. 131–139. URL: http://ceur-ws.org/Vol-956/S5_Paper1.pdf (visited on 03/01/2013).
- [2] Jöran Pieper. "Learning software engineering processes through playing games." In: *2012 2nd International Workshop on Games and Software Engineering (GAS)*. Zurich, Switzerland, Sept. 2012, pp. 1–4. DOI: 10.1109/GAS.2012.6225921. URL: <http://dx.doi.org/10.1109/GAS.2012.6225921>.
- [3] Jöran Pieper. "Die Essenz des Software Engineering – spielerisch und integriert." In: *Tagungsband des 14. Workshops "Software Engineering im Unterricht der Hochschulen" 2015*. Vol. 1332. Dresden, Deutschland: CEUR Workshop Proceedings (CEUR-WS.org), Feb. 2015, pp. 41–52. DOI: urn:nbn:de:0074-1332-7.
- [4] Jöran Pieper. "Discovering the essence of Software Engineering - an integrated game-based approach based on the SE-MAT Essence specification." In: *2015 IEEE Global Engineering Education Conference (EDUCON)*. Mar. 2015, pp. 939–947. DOI: 10.1109/EDUCON.2015.7096086.

ACKNOWLEDGMENTS

I would like to thank my university and my faculty for giving me the opportunity to start this project and bring it to a successful conclusion. Thanks to all my colleagues who supported me. In particular, I would like to thank the IACS for the possibilities of exchanging ideas and supporting the presentation of my ideas and results at colloquia, workshops, and conferences. Essential impulses for my work resulted from these events.

I would like to thank my mentor Prof. Oliver A. Lüth for his confidence, the suggestions and constructive criticism, the encouragement and the freedom he has given me in pursuing my ideas for this work.

My thanks go to my supervisor Prof. Peter Forbrig, who did not hesitate to support me with patience, confidence and constructive criticism. Thank you for the inspiring conversations.

I would like to thank Prof. Michael Goedicke for taking over the evaluation of my work without hesitation and for giving me insights into SEMAT Essence.

My sincerest thanks go to my parents for their continued support and for giving me the freedom to grow by making my own choices.

Last but not least, I would like to thank all my friends, my whole family and especially my wife for their support during my promotion. Thank you for your fantastic support. Thank you for your patience and trust in me. Siiri, you are the greatest! My dear children, you two are the best!

I'm so thankful that you're all here!

CONTENTS

1	INTRODUCTION	1
1.1	Challenges in Software Engineering	1
1.2	Challenges in Software Engineering Education	3
1.3	Goals of this Research / Gaps to Close	7
1.4	Research—Methodology, Questions, and Hypotheses	9
1.5	Outline	12
2	BACKGROUND: KNOWLEDGE AREAS AND FIELDS OF RE- SEARCH INVOLVED	15
2.1	Models Everywhere	16
2.2	Learning Theories	19
2.2.1	Behaviorist Learning Theory	19
2.2.2	Cognitivist Learning Theory	21
2.2.3	Constructivism	24
2.2.4	Beyond Constructivism	30
2.2.5	Two Instructional Design Theories	31
2.2.6	Conclusions	36
2.3	Learning Objectives And Competencies	38
2.3.1	A Taxonomy for Learning, Teaching, and As- sessing: A Revision of Bloom’s Taxonomy of Educational Objectives	38
2.3.2	Demanded Software Engineering (SE) Compe- tencies	40
2.3.3	Existing Software Engineering (SE) Curriculum Guidelines	46
2.3.4	Conclusions	51
2.4	Digital Game-Based Learning (DGBL)	55
2.4.1	Terminology	55
2.4.2	Games And Learning	59
2.4.3	Employment examples	64
2.4.4	Typology of Serious Games	64
2.4.5	Challenging Evaluation	65
2.4.6	Conclusions	67
2.5	DGBL in SE Education—Existing Solutions	68
2.5.1	Existing Simulations and Games for SE Process/ Method Education	71
2.5.2	Evaluation	79
2.5.3	Conclusions	80
2.6	Software Processes, Methods, and Practices	83
2.6.1	Towards SE Practices Instead Of Software Pro- cesses	88
2.6.2	Meta-Models for Standardized Process Descrip- tion	91

2.6.3	Conclusions	92
2.7	SEMAT Essence—A Standard for Software Engineering Methods	95
2.7.1	Kernel and Language for Software Engineering Methods (Essence)	96
2.7.2	SEMAT Essence Kernel	97
2.7.3	Essence In Education	108
2.7.4	Conclusions	108
2.8	Software Process Simulation Modeling (SPSM)	109
2.8.1	Motivation/Purpose/Objectives/Scope	110
2.8.2	Prevalent Simulation Formalisms	112
2.8.3	Conclusions	115
3	A NEW SIMULATION APPROACH SUPPORTING DGBL IN SE METHODS EDUCATION	117
3.1	Experimental Frame	117
3.2	Guiding Principles for a new Modeling Approach	117
3.3	Building interactive instructional models based on SEMAT Essence	120
3.4	DEVS and Finite State Machines	131
3.4.1	Discrete Event System Specification (DEVS)	131
3.4.2	Classic Atomic DEVS	131
3.4.3	Classic DEVS Coupled Models	133
3.4.4	Parallel DEVS	134
3.4.5	Hierarchical Models	135
3.4.6	Finite State Machines (FSM)	136
3.5	Model Description	136
3.5.1	Complexity and Level of Detail	136
3.5.2	Hierarchical Parallel DEVS Model With Ports	137
3.5.3	Model Elements Overview	143
3.5.4	Complexity Considerations	168
3.5.5	From Essence Kernel To Essence Method	169
3.5.6	Limits of Chosen Approach	170
3.6	Implementation	171
3.7	Streamlining the Modeling Workflow	172
3.7.1	Designing an Essence Kernel/Method	172
3.7.2	Making Interdependencies Explicit	172
3.7.3	Quantifying the Model	172
3.7.4	Modeling Feedback for Usage in Game Environment	175
3.7.5	Game Scenario Narrative	176
3.8	Differences to Existing Modeling Approaches	176
3.9	Future Work	177
4	AN INTEGRATED APPROACH TO LEARN SOFTWARE ENGINEERING METHODS	179
4.1	The Case for an Integrated Approach	179

4.1.1	Facilitating NOT Replacing Existing (Project-Based) Learning Approaches	182
4.1.2	Learning Theories And Educational Approaches Applied	182
4.1.3	Learning Objectives and Learner Profiles	186
4.2	Based on SEMAT Essence	189
4.2.1	Experiences With Utilizing Essence in SE Education	189
4.2.2	Essence's Support of Learning Theories And Educational Approaches	191
4.2.3	Essence's Support of Curriculum Guidelines	193
4.2.4	Conclusions	201
4.3	The Phases of The Integrated Approach	202
4.3.1	Phase 1: Getting Familiar	202
4.3.2	Phase 2: Practicing Virtually	203
4.3.3	Phase 3: Practicing in Real Projects	205
4.3.4	Phases 3+x: From Method Consumer to Method Producer	206
4.4	Integration into a Curriculum	206
4.5	Essence Kernel Puzzler	207
4.5.1	Learning Objectives	208
4.5.2	Target Audience / Player Profile	208
4.5.3	Concept and User Interface Overview	208
4.5.4	Levels	209
4.5.5	Results	209
4.6	Essence Navigator	217
4.6.1	Guiding Principles	217
4.6.2	User Interface Overview	218
4.6.3	Essence Kernel/Method Composition And Enactment	221
4.6.4	Related Work	224
4.6.5	Future Work	225
4.7	Simulation Game	226
4.7.1	Learning Objectives	226
4.7.2	Guiding Principles	227
4.7.3	Gameplay and User Interface Overview	231
4.7.4	Scoring, Ranking, Leaderboards	237
4.7.5	Teamwork—Collaboration and Competition	239
4.7.6	Feedback and Guidance	240
4.7.7	Where Are the Levels and Game Badges?	241
4.7.8	Lecturers Point of View	243
4.7.9	Debriefing	244
4.7.10	Architecture	245
4.7.11	Classification of Taken Approaches	245
4.7.12	Future Work	246
4.8	Related Work	246

5	APPLICATION AND EVALUATION	251
5.1	Case Study Settings	251
5.1.1	Characteristics of Participating Groups	251
5.1.2	Procedure of Group 1	254
5.1.3	Procedure of Group 2	255
5.1.4	Why Is There No Comparative Experiment?	255
5.1.5	Why Are There No Pre-Test/Post-Tests?	256
5.1.6	Questionnaire Design	256
5.1.7	Statistical Hypothesis Testing	257
5.1.8	Heuristics	257
5.2	Results	258
5.2.1	Essence Kernel Puzzler	258
5.2.2	Simulation Game	262
5.2.3	Essence Navigator	276
5.2.4	SEMAT Essence	281
5.2.5	Integrated Approach	281
6	SUMMARY	287
6.1	Contributions	288
6.2	Summary of Research Questions/Hypotheses	289
6.3	Future Work	292
	BIBLIOGRAPHY	295
A	CODING SCHEME OF ESSENCE KERNEL ELEMENTS USED IN GRAPHS	325
A.1	Alpha Identifiers	325
A.2	Alpha State Identifiers	325
A.3	Checkpoint Identifiers	327
A.4	Activity Space Identifiers	327
B	ANALYSIS OF ALPHAS' INTERDEPENDENCIES IN THE SE- MAT ESSENCE KERNEL—MAKING IMPLICIT DEPENDEN- CIES EXPLICIT	329
B.0.1	Alpha: Stakeholders	330
B.0.2	Alpha: Opportunity	334
B.0.3	Alpha: Requirements	338
B.0.4	Alpha: Software System	343
B.0.5	Alpha: Team	347
B.0.6	Alpha: Work	351
B.0.7	Alpha: Way of Working	355
B.1	Alignment of Analysis Results	359
C	HEURISTICS	361
C.1	Dondi and Moretti (2007) quality criteria and their map- ping to Sim4SEEd's approaches and design decisions	361
C.2	Evaluating Mapping to SE Education Requirements by Boehm (2006)	366
C.3	Evaluating Mapping to Recommendations of Jiang et al. (2015)	366

D	STATISTICAL HYPOTHESES TESTING	369
D.1	Hypothesis Test for Q09	370
D.2	Hypothesis Test for Q19	371
D.3	Hypotheses Tests	372
E	QUESTIONNAIRE USED FOR CASE STUDY	373
F	LITERATURE REVIEW—STUDIES OF SIMULATION AND DG- BL IN SOFTWARE ENGINEERING EDUCATION AFTER 2013	377
F.1	Data Sources and Search Strategy	377
F.2	Results	378
G	SELBSTÄNDIGKEITSERKLÄRUNG	383
H	RÉSUMÉ	385
I	THESIS STATEMENTS	387

LIST OF FIGURES

Figure 2.1	Intersection of Knowledge Areas	16
Figure 2.2	Relationships between Knowledge Areas . . .	17
Figure 2.3	Zone of Proximal Development (Vygotsky) . .	26
Figure 2.4	DGBL, Serious Games and Related Concepts [adapted from 40, 41]	59
Figure 2.5	Relationship Between Fun And Learning: Dif- ferent Theoretical Assumptions [adapted from 229]	62
Figure 2.6	Paradigms of Entertainment-Education [<i>adapted from 229</i>]	62
Figure 2.7	SimSE[174–176, 180–182]	76
Figure 2.8	SEMAT Essence: Method Architecture[188] . .	96
Figure 2.9	SEMAT Essence: Method Composition	97
Figure 2.10	SEMAT Essence: Kernel Made Tangible by Pro- vided Cards[280]	99
Figure 2.11	SEMAT Essence Kernel: <i>Opportunity</i> Alpha with States and Checkpoints for the <i>Identified</i> Alpha State	102
Figure 2.12	SEMAT Essence Kernel: Alphas and Their As- sociations [Screenshot of Essence Kernel Puz- zler (cf. section 4.5) adapted from 119, 188] . .	102
Figure 2.13	SEMAT Essence: Kernel, Practice, and Method Content[188]	103
Figure 2.14	SEMAT Essence: Need for Practices to Extend the Kernel[118]	105
Figure 2.15	SEMAT Essence: Two of Development Exten- sion’s sub-Alphas: <i>Bug</i> and <i>System Element</i> in- hibiting and driving, respectively, the <i>Software System</i> Alpha	106
Figure 2.16	SEMAT Essence Kernel: Plan-Do-Check-Adapt Cycle[118]	107
Figure 3.1	Alpha Progressing of the Three Areas of Con- cern	121
Figure 3.2	Progressing and Potential Retrogressing of the TEAM Alpha	122
Figure 3.3	Alpha Progressing of the three Areas of Con- cern	123
Figure 3.4	Examples of n:m Associations Between Alpha States and Activity Spaces	124
Figure 3.5	Avoiding Circular Dependencies (Direct Depen- dency)	126

Figure 3.6	Avoiding Circular Dependencies (Indirect Dependency)	127
Figure 3.7	Alpha State Interdependency Graph Used in the Simulation Model	130
Figure 3.8	DEVS Hierarchical Models	135
Figure 3.9	FSM ActivitySpace	147
Figure 3.10	I/O Ports of Alpha DEVS Model	148
Figure 3.11	I/O Ports of AlphaState DEVS Model	149
Figure 3.12	FSM Alpha State	151
Figure 3.13	I/O Ports of Checkpoint DEVS Model	153
Figure 3.14	FSM Checkpoint DEVS Model	156
Figure 3.15	I/O Ports and Input/Output Coupling of MultipleActivityCheckpoint DEVS Model	161
Figure 3.16	Connection of Atomic DEVS Models Calendar-Time and CostCalculator	161
Figure 3.17	I/O Ports of AssignmentAllocator DEVS Model	162
Figure 3.18	I/O Ports of SimPause DEVS Model	163
Figure 3.19	I/O Ports of SinglePerformanceFactor DEVS Model	163
Figure 3.20	TotalPerformanceFactor DEVS Model	166
Figure 3.21	I/O Port and HTTPS Interface of UiHttpSender DEVS Model	167
Figure 3.22	I/O Ports of IntelligentTutor DEVS Model	168
Figure 4.1	Screenshot of the Course Environment: Tools Supporting Phases 1 – 3	181
Figure 4.2	Screenshots of Puzzler’s status bar Elements	209
Figure 4.3	Screenshots of Puzzler’s Levels and Highscore	211
Figure 4.4	Total Number of Attempts by Puzzler Level	212
Figure 4.5	Successful Level Attempts of Single Players	213
Figure 4.6	Failed Level Attempts of Single Players	214
Figure 4.7	Seconds Needed to Master Puzzler Levels (outliers ($x x \in X, x > \mu_X + 4\sigma_X$) removed)	215
Figure 4.8	Seconds Before Failing a Puzzler Level (outliers ($x x \in X, x > \mu_X + 4\sigma_X$) removed)	216
Figure 4.9	Screenshot of the Essence Navigator showing Alphas Overview	219
Figure 4.10	Screenshots of the Essence Navigator Showing Alpha Balance Indicators	220
Figure 4.11	Screenshots of the Essence Navigator: Alpha and AlphaState Details of the Requirements Alpha	221
Figure 4.12	Screenshot of the Essence Navigator showing Activities Overview	222
Figure 4.13	Screenshot of the Essence Navigator showing Competencies Overview	222

Figure 4.14	Screenshot of the Essence Navigator Showing a To-Do List Based On Endeavor's Current State	223
Figure 4.15	Essence Method Parsing and Enactment Schematic Overview	224
Figure 4.16	Screenshot of the <i>SematAcc</i> [98]: GUI	225
Figure 4.17	Screenshot of Simulation Game: UI Overview	232
Figure 4.18	Screenshot of Simulation Game: Analyzing Feedback via Alpha Assessment	233
Figure 4.19	Screenshots of Simulation Game: Color Coding of Messages	234
Figure 4.20	Screenshots of Simulation Game: Exploration of Activities Supported By Navigator	235
Figure 4.21	Screenshots of Simulation Game: Assigning Activities To the Virtual Team	235
Figure 4.22	Screenshot of Simulation Game: Incoming Messages of Virtual Team	235
Figure 4.23	Screenshot of Simulation Game: Time Travel	237
Figure 4.24	Screenshot of Simulation Game: Ranking Stats Presented In the status bar	238
Figure 4.25	Screenshots of Simulation Game: Leaderboards	239
Figure 4.26	Screenshot of Simulation Game: Status Bar Details	241
Figure 4.27	Screenshots of Simulation Game Results: Alpha Assessment Comparison	241
Figure 4.28	Screenshots of Simulation Game Results: Team Performance Comparison	242
Figure 4.29	Screenshots of Simulation Game Results: Check-point Correctness	242
Figure 4.30	Screenshots of Simulation Game Results: Cost and Score	242
Figure 4.31	Screenshots of Simulation Game: Confirmation Message and Tutor Message	242
Figure 4.32	Components Architecture	245
Figure 5.1	Q01: Do you Play Digital Games?	252
Figure 5.2	Q03: How much time do you spend playing games per week?	253
Figure 5.3	Q06: Do you think games provide learning?	253
Figure 5.4	Q07: Did you already work on software projects outside of your curriculum?	254
Figure 5.5	Q31: Did you use the Puzzler to be prepared for the Simulation Game?	258
Figure 5.6	Q32: How much time did you spend with the Puzzler?	259

Figure 5.7	Q33: How much did the Puzzler help you to get familiar with Essence concepts and vocab?	260
Figure 5.8	Q34: How difficult was the Puzzler for you?	260
Figure 5.9	Q35: Will you use the Puzzler again to reinforce your Essence Kernel vocab?	261
Figure 5.10	Q36: Did the Puzzler prepare you well for the Simulation Game?	261
Figure 5.11	Q37: Would you recommend the Puzzler to a friend/fellow student?	262
Figure 5.12	Game Results of Teams In the Case Study	263
Figure 5.13	Q09: How much fun did you have playing the Simulation Game?	268
Figure 5.14	Q10: How difficult was the Simulation Game for you?	269
Figure 5.15	Q11: How was the duration of the Simulation Game in your opinion?	270
Figure 5.16	Q17: Would you play the Simulation Game again?	271
Figure 5.17	Q19: Would you recommend this game to a friend or fellow student?	271
Figure 5.18	Q20: Should this game be a standard part of any SE course?	272
Figure 5.19	Q22: Did you learn something about SE by playing the Simulation Game?	272
Figure 5.20	Q23: Did playing the Simulation Game reinforce your SE knowledge of prior lectures/-courses?	273
Figure 5.21	Q25: Did you learn something new about SE by playing the Simulation Game?	273
Figure 5.22	Q27: Did you improve your performance (higher score, less costs) while playing the Simulation Game?	276
Figure 5.23	Q42: How familiar was SEMAT Essence to you BEFORE the Simulation Game?	277
Figure 5.24	Q43: How familiar was SEMAT Essence to you AFTER the Simulation Game?	277
Figure 5.25	Q39: Would you like to deploy the Essence Navigator in one of your future projects?	280
Figure 5.26	Q40: Should a tool like the Navigator be standard in any SE project course?	281
Figure 5.27	Q44: Do you think SEMAT Essence and the Essence Kernel will be of help in your future projects?	282
Figure 5.28	Q45: Do you want to deploy SEMAT Essence in your future SE projects?	282

LIST OF TABLES

Table 1.1	Project resolution results from CHAOS research for years 2004 to 2012 [281, 282]	2
Table 1.2	Time and cost overruns, plus percentage of features delivered from CHAOS research for the years 2004 to 2012. [281, 282]	3
Table 2.1	Summary of major learning theories	20
Table 2.2	Cognitive Load Theory: Distinctions Between biologically primary and secondary knowledge [195, examples added]	32
Table 2.3	Cognitive Load Theory: The 3 Categories of Cognitive Load [adapted from 195]	34
Table 2.4	Revised Taxonomy: Knowledge Dimension[12] 41	
Table 2.5	Revised Taxonomy: Cognitive Process Dimension[12]	42
Table 2.6	Taxonomy Table of The Revised Taxonomy[12]	43
Table 2.7	SEEK Knowledge Areas[133]	47
Table 2.9	GSwE2009: Knowledge Areas And Knowledge Units[269]	52
Table 2.10	GSwE2009: Additional Expected Outcomes[269] 53	
Table 2.11	(Systematic) Literature Reviews of Simulation and DGBL in SE Education (adapted from Jiang et al.[126])	68
Table 2.12	Timeline of Simulation-Based DGBL Approaches in SE Education Focused On SE Process/Methods	70
Table 2.13	Existing DBGL Approaches In SE Education .	72
Table 2.14	Agile and Plan-Driven Method Home Grounds [adapted from 30]	87
Table 2.15	Problems Caused by Current Generation of Software Processes[122]	90
Table 2.16	Comparison of Industry Standards for Standardized SE Process Description	93
Table 2.17	SEMAT Essence Kernel: Elements By Area Of Concern	98
Table 2.18	SEMAT Essence Kernel: Customer Area of Concern—Alphas[188]	100
Table 2.19	SEMAT Essence Kernel: Solution Area of Concern—Alphas[188]	101
Table 2.20	SEMAT Essence Kernel: Endeavor Area of Concern—Alphas[188]	101

Table 2.21	Comparison of Modeling Techniques: <i>System Dynamics</i> vs. <i>Discrete Event System</i> [139, 311] . . .	114
Table 3.1	List of Models	144
Table 3.2	I/O Ports of ActivitySpace DEVS Model . . .	146
Table 3.3	I/O Ports of Alpha DEVS Model	148
Table 3.4	I/O Ports of AlphaState DEVS Model	150
Table 3.5	Messages sent by Checkpoint DEVS Model to Provide Feedback in an Interactive Environment	154
Table 3.6	I/O Ports of Checkpoint DEVS Model	155
Table 3.7	I/O Ports of MultiActivityCheckpoint DEVS Model	158
Table 3.8	I/O Ports of MultiActivityCheckpointAllocator DEVS Model	159
Table 3.9	I/O Ports of MultiActivityCheckpointAggregator DEVS Model	160
Table 3.10	I/O Ports of CalendarTime DEVS Model . . .	160
Table 3.11	Examples of SinglePerformanceFactors Used in Case Study Model	164
Table 3.12	Tutor Message Examples	167
Table 3.13	Quantifying the Simulation Model—Necessary Model Input	173
Table 4.1	<i>Integrated Approach</i> Addressing Jonassen's _[134] Characteristics of Constructivist Learning Environments Facilitating Learning	183
Table 4.2	Elements Of the Integrated Approach Addressing Gagnè's Events Of Instruction	185
Table 4.3	Learning Objectives of Phases of the Integrated Approach Mapped to the Revised Bloom's Taxonomy[13, 147]	188
Table 4.4	3+X Phases of the Integrated Approach	202
Table 4.5	Short Description of Puzzler's Levels	210
Table 4.6	Game-Based Learning for Universities and Life Long Learning classification[74] (selection) . .	230
Table 4.7	Games Classification By Labels[40, 41]	247
Table 5.1	Game Results of Teams In the Case Study . . .	264
Table 5.2	Utilization of the Time Travel Feature by Teams	266
Table 5.3	Q28: What helped you the most to improve your performance while playing the Simulation Game?	278
Table 5.4	Q29: What helped you the most to learn while playing the Simulation Game?	279
Table 5.5	Summary of Hypotheses Testing Regarding Features of the Simulation Game obtained by Sign-tests	280

Table A.1	Alpha Identifier Coding Scheme Used in Graphs	325
Table A.2	Alpha State Identifier Coding Scheme Used in Graphs	325
Table A.2	Alpha State Identifier Coding Scheme Used in Graphs	326
Table A.2	Alpha State Identifier Coding Scheme Used in Graphs	327
Table A.3	Activity Space Identifier Coding Scheme Used in Graphs	328
Table C.2	Special Interest Group for Game-Based Learning in Universities and Lifelong Learning Quality Criteria[74] and Their Mapping to Taken Approaches and Design Decisions	365
Table C.3	Evaluating Mapping of Approaches Taken to Recommendations of Jiang et al.[126]	367
Table D.1	Hypothesis Tests Summary	372
Tabelle F.1	Study Selection By Data Sources	378
Tabelle F.2	Identified Simulation and DGBL Studies In Software Engineering Education Focussed on SE Process/Method	378
Tabelle F.2	Identified Simulation and DGBL Studies In Software Engineering Education Focussed on SE Process/Method	382

ACRONYMS

ACM Association for Computing Machinery

AI Artificial Intelligence

CBOK Core Body of Knowledge

CIP Cognitive Information Processing

CLT Cognitive Load Theory

CMMI-DEV Capability Maturity Model Integration for Development

CORBOK Core Body of Knowledge

CS Computer Science

DES Discrete Event Simulation

DEVS Discrete Event System Specification

DGBL	Digital Game-Based Learning
EPF	Eclipse Process Framework
ERP	Enterprise Resource Planning
FSM	Finite State Machine
GSD	Global Software Development
IEEE	Institute of Electrical and Electronics Engineers
ITS	Intelligent Tutoring System
KA	Knowledge Area
MAC	Mind as Computer
MKO	More Knowledgeable Other
MOOC	Massive Open Online Course
OMG	Object Management Group
PBL	Problem-Based Learning
PMBOK	Project Management Body of Knowledge
RCT	Randomized Controlled Trial
RMC	Rational Method Composer
SD	System Dynamics
SE	Software Engineering
SEEK	Software Engineering Education Knowledge
SLR	Systematic Literature Review
SPEM	Software Process and Systems Engineering Meta-Model
SPI	Software Process Improvement
SPM	Software Process Modeling
SPS	Software Process Simulation
SPSM	Software Process Simulation Modeling
SWA	Software Assurance
SWEBOK	Software Engineering Body of Knowledge
SWEBOS	Software Engineering Body of Skills
SWECOM	Software Engineering Competency Model

UML Unified Modeling Language

XMI XML Metadata Interchange

ZPD Zone of Proximal Development

INTRODUCTION

1.1 CHALLENGES IN SOFTWARE ENGINEERING

The last three decades brought many-faceted technological changes with far-reaching influences on how we shape our lives—personally, socially and relating to business. Those developments changed the way how we work, how we spend our time and how we interact with our environments. They changed fundamentally the way and the speed we access information and how companies shape their business processes. In the year 2013 for the first time, the annual sales of smartphones surpassed sales of so-called dumbphones.[91] The incredible computing power of multi-core processors people today wear in their pockets, and the capabilities to interact with it via voice and gestures were inconceivable only a few years ago.

Aside from all big breaks in hardware development the innovations are driven by software. Finally, it is software that makes devices and services smart.

On the downside, all those technological breakthroughs were accompanied by a high ratio of failing projects, causing immense damages—economically and in extreme cases endangering human lives. Most notably perceived were IT projects of large scale that failed or got challenged, such as the failed first flight of an Ariane 5 rocket in 1996 causing a loss of hundreds of millions of US dollars, the delayed initial operation of TollCollect, a HGV toll system in Germany, which caused billions of Euro in lost revenues[22] and long-term problems with HealthCare.gov that inhibited American to find and negotiate health insurance contracts. Almost every year data and security leaks get revealed, compromising or inadvertently exposing private data of several million people, such as happened to Adobe[109] and Facebook in 2013[253] to name just a few.

In 1994 the often cited *Standish Chaos Report* [281], announced that only 16.2% of all software projects were finished successfully. The report claimed that 52.7% of the projects were challenged (did not meet budget, time and features specified) and 31.1% of the projects were canceled during the development. It stated that the average cost overrun was 189% of original cost estimates, that the average time overrun of challenged and impaired projects would amount to 222% of the original time estimate and that on average only 61% of originally specified features were available on challenged projects.[281] These statements were regularly cited as an evidence of a *software crisis* and were largely used as a reference by IT departments, sci-

“Engineering is all about selecting the most appropriate method for a set of circumstances.”

—Ian Sommerville
(2010)

	1994	2004	2006	2008	2010	2012
Successful	16.2%	29%	35%	32%	37%	39%
Failed/Canceled	52.7%	18%	19%	24%	21%	18%
Challenged	31.1%	53%	46%	44%	42%	43%

Table 1.1: Project resolution results from CHAOS research for years 2004 to 2012 [281, 282]

entific researchers, software consulting companies and government advisors.[84, 136] It was claimed that software projects, in general, are unreliable and likely to be over budget and behind schedule.[96]

The figures of the Standish Reports, their associated statements, as well as their broad acceptance and citations, are questioned by software development practitioners and researchers¹ but provide a comprehensive observation over a long period.

Table 1.1 shows raising success figures of the last decade for projects in the periodically appearing Standish Reports compared to the original figures from 1994. According to these figures in 2012 were 39% of the projects successful by report's definition—meaning that the projects were finished in estimated schedule and budget and delivered with specified features. With 18% the number of canceled projects is significantly lower than the figure of 1994. 43% of the projects were reported with overruns in time and budget or with less features than originally specified. As table 1.2 shows, the overruns reported by the Standish Reports declined as well, which could show that projects performed better and/or their budget and schedule estimates were made more cautious or more accurate at best.

The numbers of the late reports align better to reported dimensions of other comprehensible and objective research studies. Those studies reported a combined rate of canceled plus not successful software projects to be between 26% and 34% (in years 2005 and 2007) and a cancellation rate ranging from 11.5% to 15.5%.[84] Earlier cost overrun surveys from 1984, 1988 and 1992 “suggest an average cost overrun in the range of about 30%.”[136] Thereby it has to be mentioned that the figures can't be compared directly to those of the Standish Reports because the measures were defined differently to some extent.

Looking at these numbers, it may be exaggerated to talk of a software crisis, where the majority of projects get canceled, but by “[...] most standards, this would be considered a high failure rate for an

¹ Criticisms include findings, as well as the methodology of the Standish Group reports[96].

	1994	2004	2006	2008	2010	2012
Time overrun	222%	84%	72%	79%	71%	74%
Cost overrun	89%	56%	47%	54%	46%	59%
Features delivered	61%	64%	68%	67%	74%	69%

Table 1.2: Time and cost overruns, plus percentage of features delivered from CHAOS research for the years 2004 to 2012. [281, 282]

applied discipline.”[80]

According to Sommerville[260], key challenges for software developers today and tomorrow are increasing diversity and the need for shortened delivery times while guaranteeing trustworthy quality. Software Engineering (SE), defined as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software [...]”[112], is required to meet those demands.

While it is obvious that Software Engineering (SE) education can and will not eliminate all the reasons for failing software projects, it significantly shapes the perspectives taken by tomorrow’s software engineers.

1.2 CHALLENGES IN SOFTWARE ENGINEERING EDUCATION

The development of complex and highly interconnected software systems demands well-educated software engineers, capable of choosing the right tools and methods to accomplish dynamic requirements. Thereby different scenarios require to consider various aspects, to set different priorities and eventually to select proper tools, practices, and methods.

SE education is challenged in many ways. It has to take enormous diversity into account and to offer a well-founded range of knowledge to enable tomorrow’s software engineers.

Conditioned by limited time, all contents taught are just a selection out of a huge, diverse spectrum. While depth and breadth of selected contents vary in detail, it is broad consent, that beside all technological aspects and tools a profound knowledge about *software processes*, which describe approaches to the production and evolution of software, are crucial for successful applications of software engineering.

Emerging agile and lean approaches propagate the renouncement of ballast—the disuse of methods or tools, which form the foundation

of more traditional development processes. Lacking experience in developing complex software systems, the consequences of (not) using methods and tools can be judged only hardly and unrefinedly.

The Software Engineering Body of Knowledge (SWEBOK) V3.0 [112] subdivided the field of software engineering into fifteen knowledge areas:

- *software requirements,*
- *software design,*
- *software construction,*
- *software testing,*
- *software maintenance,*
- *software configuration management,*
- *software engineering management,*
- *software engineering process,*
- *software engineering models and methods,*
- *software quality,*
- *software engineering professional practice,*
- *software engineering economics,*
- *computing foundations,*
- *mathematical foundations and engineering foundations* supplemented by an additional knowledge area called
- *related disciplines, including*
 - *computer engineering,*
 - *computer science,*
 - *general management,*
 - *mathematics,*
 - *project management,*
 - *quality management, and*
 - *systems engineering.*

While some of those knowledge areas are relatively well suited to be learned by lectures and associated laboratory tutorials, others are less eligible. Those include *software engineering process* as well as *software engineering models and methods*².

² including *Agile Methods*, cf. section 2.6

“Anyone who has tried to teach topics such as [...] process, [...] will recognize the glassy-eyed appearance in the eyes of some (or most) students. These are critical topics for industrial practice, yet it is a particular challenge to motivate students to feel passionate in these areas, and hence learn what they need to know.”[152] Timothy[286] states “[...] there is a clear knowledge gap and a reliance on on-the-job learning for topics related to software processes [...].” He adds “[...] the greatest on-the-job learning occurs in the software process category [...]”[286] and “[...] the large amount of on-the-job learning—and greater importance relative to amount known—suggest that educational institutions should place considerably more emphasis on teaching topics such as [...] software processes [...].”[286]

Lethbridge[152] asks “How can process topics be taught better? What types of case studies, simulations or other exercises will work most effectively?” Class or capstone projects controlled by the lecturer are a common supplement to lectures. General circumstances in an academic setting, e.g. limited time and other resources as well as the demand to assess individual performance of students, are constraining size, scope, and type of such projects. To enable positive project experiences and successful project realization, lecturers limit scopes, preselect appropriate technology, tools, methods and processes that have to be followed by students in their projects.

Growing the size of project teams, expanding communication demands, increasing the division of responsibilities and the insistence on the delivery of a fixed project outcome increase the degree of reality.

But in a course or capstone team project, not every project member can fill the role of a project manager, project owner, etc. to feel the difficulty of handling conflicting goals.

Project members often just deliver project’s artifacts and lose a holistic view of the software process as a whole.

Limited time doesn’t offer the ability to test different alternative strategies or other software development life-cycles.

Sommerville emphasized that “engineering is all about selecting the most appropriate method for a set of circumstances.”[260] To enable conscious decisions of tomorrow’s software engineers, they have to know their options and to be able to take multiple aspects and perspectives into account. At this point, SE education has to provide concepts for retaining a holistic view on the whole SE endeavor while working focused on the required details at the same time.

Boehm[27] reviewing SE in the 20th and 21st century concludes that SE education is required to

- “Keeping courses and courseware continually refreshed and up-to-date;”
- “Anticipating future trends and preparing students to deal with them;”

- “Monitoring current principles and practices and separating timeless principles from out-of-date practices;”
- “Packaging smaller-scale educational experiences in ways that apply to large-scale projects;”
- “Participating in leading-edge software engineering research and practice, and incorporating the results into the curriculum;”
- “Helping students learn how to learn, through state-of-the-art analyses, future-oriented educational games and exercises, and participation in research;”

Shaw[251] explains: “Engineering entails creating cost effective solutions to practical problems by applying scientific knowledge, building things in the service of mankind. Engineers preferentially apply scientific and mathematical knowledge when it’s available and rely on less systematic knowledge at other times. Engineers work under limitations of both time and knowledge. They are responsible for reconciling conflicting constraints, especially cost constraints. Engineers make deliberate choices among alternative designs for both technical and nontechnical reasons. Their judgments are based on deep knowledge of the discipline in which they design, and they assume personal responsibility for the safety and quality of the systems they design. [...]” She proposes to “Present theory and models in the context of practice: Emphasize durable ideas that will transcend a major shift of technology. Students often learn them best when they appear in concrete examples; good examples will themselves be worth remembering for reuse.”

To meet those requirements, SE education has to provide a holistic view on SE endeavors—one that is transferable to different contexts and students’ upcoming challenges.

1.3 GOALS OF THIS RESEARCH / GAPS TO CLOSE

This research work aims at closing different gaps identified. Its goal is to provide an environment which fosters learning of SE practices and SE methods in a way that enables students to realize and feel that these concepts give real support and orientation in the multidimensional process of software development where so many incoming information have to be dealt with in an appropriate way.

Missing excitement on the part of students when it comes to software process should come as no real surprise. Without experience building more complex software systems the need for such disciplined approaches may not be as obvious as desired. A perception that (trivial) projects in the past were more or less manageable without ‘restricting rules’ is widely spread. Discussion, deeper analysis or comparison of SE practices and methods in the form of a dialog, which would be beneficial to learning, is complicated by the fact that students often cannot draw on any own experiences. So they would have to take it for granted—or not. In course or capstone projects the following observations can be made:

- It is not easy for students to orientate inside of a given SE method or software process.
- It is quite hard for students to answer the question(s) who, how, when and in particular *why* specific activities should be accomplished.
- The reasonable division of responsibilities inside of student project teams lets their members focus on delivering demanded artifacts leading to a quick specialization of team members. A rigid fixation on technological and functional details makes them lose the holistic view on the development process as a whole—with all its relevant dimensions.
- All too frequent takeaways from such projects are too specific and hardly transferable to other contexts and future challenges.

It can be observed that students are leaving their projects with a fuzzy impression that applying SE methods/practices/tools could have helped to reach better results if they would have been employed in a more rigorous, disciplined, and goal-oriented way. Losing a holistic view due to remarkable cognitive load and just delivering artifacts requested by a preselected software process under time pressure can hardly provide the deep impression that practices, methods, and tools were indeed supporting students’ work.

This research/work aims at providing students with tools and methods that give orientation and support for goal-oriented structured acting, providing a familiar environment already appreciated at the

“Each meaning, which I realize by myself, each rule, I establish according to my own lights, convinces me more and motivates me higher than any extrinsic meaning that I hardly comprehend.”

—Kersten Reich
(2012)

start of such projects, lowering the overall cognitive load and increasing the transferability of the knowledge gained. Instead of offering a rather isolated game-based learning activity, this research aims at delivering an *Integrated Approach* to provide transferable skills and explicitly preparing students for their real project work to enable experiences that support the development of SE attitudes.

Preceding research showed that simulation and games were able to foster empathy with the necessity of employing software processes to accomplish more complex software projects to a certain level successfully. But evaluation of those approaches showed too that they were not able to impart new knowledge.[301]

Reich[223] formulates “Each meaning, which I realize by myself, each rule, I establish according to my own lights, convinces me more and motivates me higher than any extrinsic meaning that I hardly comprehend.”³ This guideline of constructivist didactics describes an ideal result of learning and deep understanding. To foster such learning, these didactics⁴ demand a learning environment that is encouraging social interaction, e.g. articulation, reflection, collaboration and competition. Unfortunately, there seems to exist a noticeable underemployment of constructivist insights in existing DGBL approaches.

The binding of current approaches to single software processes limits their transferability to new contexts requiring a different set of SE practices. With only a few exceptions, the architecture and modeling process of underlying simulation of existing approaches prevents reuse. Modeling requires substantial initial effort that is not of direct use outside of the specific game environments. Static models do not allow for customizations. Complex simulation models impede their transparency and adaption to lecturers’ and learners’ needs, leaving instructors with some extent of uncertainty if the chosen model transfers the right lessons to be learned. This research aims at providing a transparent, mostly declarative modeling approach, where a modeler is less concerned with simulation modeling constructs but more with the relationships and interdependencies of elements of the underlying domain.

In the field of software processes, a trend towards flexible SE practices instead of monolithic software processes is noticeable (cf. section 2.6.1 on page 88). The proponents deliver strong arguments to do so, both from an SE and an SE educational perspective. With SE-MAT Essence[188] (cf. section 2.7) a new standard is available that provides a kernel and a language for SE methods, which can be uti-

³ This quote was translated from German: “Jeder Sinn, den ich selbst für mich einsehe, jede Regel, die ich aus Einsicht selbst aufgestellt habe, treibt mich mehr an, überzeugt mich stärker und motiviert mich höher, als von außen gesetzter Sinn, den ich nicht oder kaum durchschaue.”[223]

⁴ Cf. section 2.2.3 for more details about *social constructivism*.

lized to integrate any practice or process. Currently, there is a lack of engaging learning material available that promotes such new approaches. This research aims at providing concepts, approaches, and tools to close that gap to some extent as well.

This research work aims not at providing a distance learning tool or an environment. The approaches envisaged rather aim at facilitating direct social interaction assuming students and lecturers, or facilitators, to be present on location. This does not mean that concepts and tools to be developed could not be utilized in distributed learning environments—but such a setting is not on primary focus, and it is assumed that such a setup would require further customization and extension of the approaches to be taken.

This section listed gaps this research aims to close. Approaches to close them are discussed in detail in following chapters.

1.4 RESEARCH—METHODOLOGY, QUESTIONS, AND HYPOTHESES

Based on an extensive literature study, with starting points provided by latest existing meta-studies in relevant fields of research, an analysis of existing approaches and current developments in corresponding fields revealed findings, innovative approaches as well as gaps to close to achieve intended objectives.

These findings built the foundation to develop new concepts and approaches aiming to close identified gaps. Vision and initial concepts were presented very early to the SE education community at national and international conferences and workshops to collect feedback and to find agreement or disagreement.

A lack of tools supporting the developed concepts and approaches required comprehensive own development efforts to create an intended integrated approach.

A qualitative case study combined with a questionnaire provides results for evaluating findings, concepts, and tools developed to support the approach.

Research Questions and Hypotheses

The primary objective of this research work is to provide students in SE courses with *competencies*

- to *orientate* inside of an SE endeavor, in all its phases,

- to enable them to answer the questions *when*, *how* and in particular *why* specific activities in an endeavor should be accomplished, and
- to keep a holistic view on the development endeavor as a whole—with all its relevant dimensions—at working focused on details required by the endeavor.

Competencies combine *related knowledge*, *skills*, and *attitudes* enabling a person to accomplish the tasks in a given context.⁵ After an analysis of existing solutions and their evaluations a number of research questions were identified and a number of hypotheses were formulated to guide research activities. These research questions and hypotheses get introduced next.

To enable the acquisition, or construction⁶, of *knowledge* effectively, learning theory has to be applied. This raises following research questions (RQs):

- RQ01: Are concepts and approaches of constructivist learning theory⁷ already fully utilized effectively?
- RQ02: How to arrange Digital Game-Based Learning (DGBL) in SE education⁸ to create new kinds of interaction between participants and lecturers to facilitate communication, analysis, and reflection?
- RQ03: How to integrate learning material into digital SE games and guide players through it to enable the exploration of new knowledge beside already learned contents?
- RQ04: How to design and implement a highly intrinsic learning game utilizing learning theories and providing highly transferable knowledge and skills?
- *Hypothesis-1*: Constructivist learning approaches, esp. those of social constructivism, are not utilized to their full potential in existing DGBL approaches in SE education.

To provide contemporary knowledge and skills in the area of SE processes/methods the high and ever-increasing number of existing SE processes/methods as well as a demanded orientation towards the utilization of flexible and composable SE practices instead of rather monolithic SE processes have to be taken into account. This poses following research questions:

⁵ cf. section 2.3.2 on page 40 for a detailed definition of *competency*

⁶ cf. section 2.2 on page 19 for different perspectives on this subject

⁷ cf. section 2.2.3

⁸ In the context of this thesis, research questions are focused esp. on SE processes/methods.

- *RQ05*: How could the ever-increasing number of SE processes/methods encountered and highly *transferable knowledge* and *skills* provided to students of SE at the same time?
- *RQ06*: How to integrate the orientation towards an utilization of flexible composable SE practices?

A number of specifications, provided by industry efforts, is aiming to standardize the description of SE processes/methods. With SEMAT Essence[188] a new⁹ standard is emerging combining a unified description and communication of SE methods with an orientation towards flexible composable SE practices. This raises the following research questions:

- *RQ07*: Is the emerging SEMAT Essence standard appropriate to facilitate the achievement of stated learning objectives?
- *RQ08*: Could an educational simulation and DGBL approach be built on top of SEMAT Essence?
- *RQ09*: How to design a simulation approach that requires less training effort—effort that is of use outside of a particular simulation and game environment?
- *Hypothesis-2*: SEMAT Essence facilitates the achievement of stated learning objectives.
- *Hypothesis-3*: An educational simulation model as well as a highly intrinsic digital learning game can be built on top of SEMAT Essence.

With regards to SE processes/methods, students develop skills and attitudes usually by applying their knowledge in course or capstone projects that are demanded by all contemporary curriculum guidelines¹⁰. Students are facing remarkable cognitive load in course/capstone projects. This poses following research questions:

- *RQ10*: How to lower cognitive load of students in course capstone projects?
- *RQ11*: How to support students to develop a professional *SE attitude* appreciating the support provided by SE methods and tools?
- *RQ12*: How to provide better orientation and guidance inside of SE endeavors to students in course/capstone projects?

⁹ Version 1.0[187] of “Kernel and Language for Software Engineering Methods (Essence)” was published by the Object Management Group (OMG) in November 2014. The current updated revision 1.1[188] was published in December 2015. SEMAT Essence gets described in detail in section 2.7.

¹⁰ cf. section 2.3.3 on page 46

- *RQ13*: What tools (languages, concepts, tools s. str.) could provide support?
- *RQ14*: Could DGBL learning experiences (better) prepare students for their project work?
- *RQ15*: How to integrate DGBL activities into the whole context of the instructional design of a software engineering course or curriculum?
- *Hypothesis-4*: Students, who experience the support of SE methods and tools providing orientation and guidance, appreciate them and develop an attitude wanting to utilize them in future projects.
- *Hypothesis-5*: Preparational activities provided to students can decrease the cognitive load they are facing in their course project work.
- *Hypothesis-6*: Students can be provided with tools to support a holistic view on their course/capstone project right from its start.
- *Hypothesis-7*: An approach, integrating DGBL activities deeply into an SE course/curriculum, fosters students' competencies with regards to the stated learning objectives.

1.5 OUTLINE

This dissertation is structured as follows:

Chapter 1 described challenges SE and SE education are facing. The existing gaps, this research aims to close, were described as well. The chapter elaborated the approach taken in this research and formulated research questions and hypotheses to examine.

Chapter 2 familiarizes with the background of this work. Different learning theories and their implications get presented. Those theories, as well as competencies demanded by presented curriculum guidelines, are primary drivers for key decisions made in chapter 4 on page 179.

Why DGBL is well suited to support learning gets explained next. DGBL and its applications in the domain of software engineering are covered. Approaches, taken so far in this domain, get presented, and their shortcomings get discussed.

Following sections describe the domain and motivation of software processes, which have been documented in very different ways and formats. Industry efforts to standardize documentation of software processes, SE practices, and methods are presented. One of them,

SEMAT Essence, gets explained in more detail. It is a result of a demanded move towards the use of composable SE practices instead of monolithic SE processes and builds the foundation of the simulation model, which gets described in chapter 3. To provide some background for this simulation model, chapter 2 provides a short introduction to software process simulation modeling (SPSM) and introduces prevalent formalisms utilized to model software processes for simulation.

Chapter 3 introduces the guiding principles of a new simulation approach. The building blocks of a model for interactive educational simulation of SE methods are described in detail and elements of the simulation model to be developed get explained. The chapter closes with an introduction to the streamlined multi-stage modeling workflow designed to free lecturers from the need to invest high training efforts, which are of use only in a particular game environment.

Chapter 4 describes a new *Integrated Approach* for SE method education aiming at lowering the cognitive load of students in course/-capstone project work and at providing a learning experience making them *want* to utilize the methods provided.

This chapter introduces the developed *Integrated Approach* and its 3+X phases. Tools to support this approach get described next. They include a *Simulation Game* preparing students for their course/capstone project work. Collaboration, competition and the deep integration of the underlying domain into the gameplay as well as the promotion of analysis and reflection throughout the game experience are unique characteristics of this approach. With learners and lecturers and in mind these tools facilitate their social interaction and supplement different kinds of educational approaches.

Chapter 5 describes the application and evaluation of the presented *Integrated Approach* and the supporting tools included. The settings of the case study conducted as well as the results of its evaluation get presented and their implications get discussed.

Chapter 6 summarizes findings, results, and contributions made in this dissertation. It closes with suggestions for future work in this subject area.

BACKGROUND: KNOWLEDGE AREAS AND FIELDS OF RESEARCH INVOLVED

This chapter introduces the background of this research. Analyzing, designing and implementing a digital game-based learning approach, concerned with SE methods and based on simulation, is a highly interdisciplinary endeavor. Multiple fields of research and theory have to be taken into account:

1. To create an integrated game-based *learning* approach, the relevant concepts of learning theories have to be employed. Competencies to be acquired have to be known and defined in a structured and comprehensive way. That is why this chapter starts with exploring different learning theories, concepts of instructional design and a taxonomy to define intended competencies (cf. 2.2 on page 19).
2. To create a *digital game-based learning* (DGBL) approach the field of DGBL has to be explored, and preliminary findings deployed to provide a highly effective learning environment (cf. section 2.4). Existing approaches to utilize DGBL in SE education, esp. in software process education, get analyzed to identify gaps to close (cf. section 2.5).
3. To create an approach that introduces students to the world of *SE methods*, the field of software *processes*, *SE practices* and *methods* has to receive an examination. Section 2.6 on page 83 describes characteristics of software processes, the proposed transition to SE practices and gives a short introduction into industry efforts to standardize the description of software processes, respectively practices and methods. Essence, a new specification of a “kernel and language for software engineering methods”[188], gets introduced in more depth since it lays the foundation of the *Integrated Approach* presented in this thesis. Benefits provided to SE education using this new standard get explained in depth.
4. To create an approach that is based on *simulation* of SE methods, an examination of existing formalisms and approaches taken so far is needed. Section 2.8 on page 109 provides an overview of existing simulation formalisms and their qualification for the proposed integrated approach.

Figure 2.1 illustrates the knowledge areas involved. The sweet spot for the integrated approach, introduced in chapter 4 on page 179, is lo-

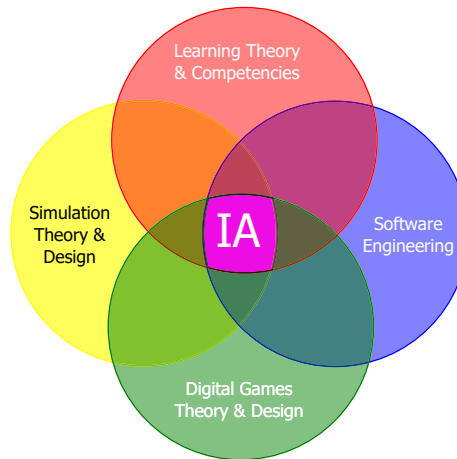


Figure 2.1: Intersection of Knowledge Areas

cated at the center of the intersection of knowledge areas—involving knowledge of all of these fields. Figure 2.2 on the next page presents the relationships between knowledge areas from the perspective of this research work. Associations between knowledge areas are presented by arrows and named from the perspective of this research.

2.1 MODELS EVERYWHERE

*“All models are
wrong.”
—John D. Sterman
(2002)*

The world of science and everyday life is full of models. And so is this thesis—we are facing models in almost all chapters of this thesis. We find models of learning theories, human cognitive models, competency models, SE, a discipline full of models, simulation, a field that is based fundamentally on models, learner models, all our mental models, even models of models (metamodels), etc..

As Ludewig[155] summarized: “We use models when we think about problems, and when we talk to each other, and when we construct mechanisms, and when we try to understand phenomena, and when we teach. In short, we use models all the time,” and he adds “models have never been invented, they have been around (at least) since humans started to exist.”

With so many types and backgrounds, it’s hard, or impossible[155], to find a consistent common definition of it. For the purpose of this thesis, the following is assumed:

According to Stachowiak[265] a candidate has to provide three features to qualify as a model:

1. *Mapping*: it is based on an “*original*.”
2. *Reduction*: it only reflects a selected subset of relevance of original’s properties.
3. *Pragmatism*: it needs to be usable in place of the original for some *purpose*.

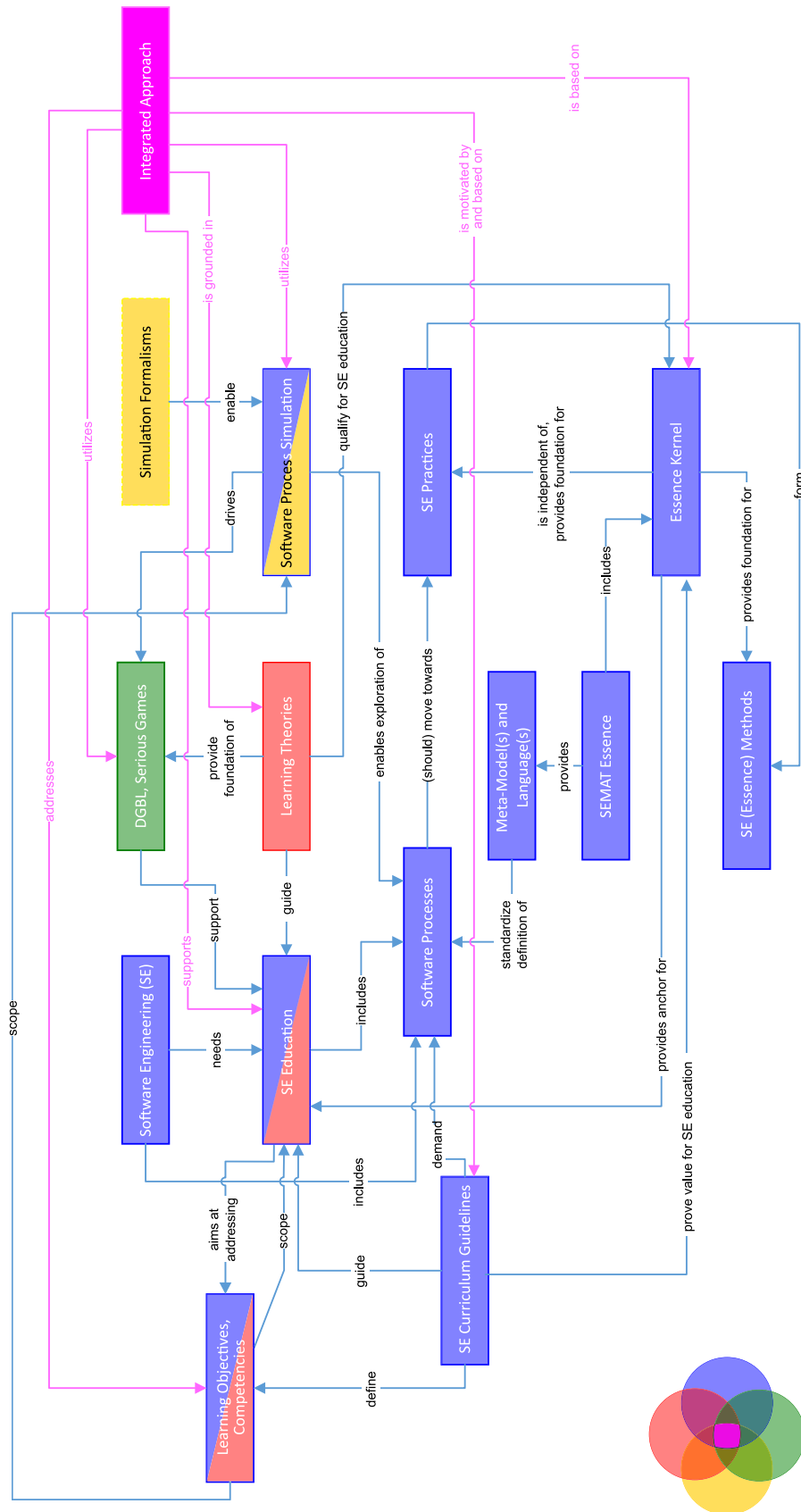


Figure 2.2: Relationships between Knowledge Areas

Mapping does not prescribe that the *original* has to be existing in reality. It might be planned, exist just conceptionally, or be completely fictional¹.

Reduction might seem like a weakness of a model, but actually, it provides its fundamental strength. With characteristics reduced to the identified relevant essentials, a model might be handled, where the *original* cannot. *Mapping* and *Reduction* in combination “lead to contradictory demands on the model, a model will always be the result of a compromise between these demands.”[294]

Pragmatism is the reason to build and use models. “Since we are not able or not willing to use the original, we use the model instead.”[155] The *purpose* is driving the design of the model. That is why “several consensus models may co-exist with respect to the same target [...] different choices may be made, resulting in the development or selection of different models. For instance, biochemists and theoretical chemists may use quite different models for the molecular structure of water as they ask different kinds of research questions.”[294]

Models may evolve over time iteratively as part of or in reaction to research activities. *Bloom’s Taxonomy of Learning Objectives* (cf. section 2.3.1 on page 38) provides such an example. Other models, most notably in times of *scientific revolutions*[148], may get discarded completely.

As Sterman² summarizes: “All decisions are based on models, and all models are wrong. These statements are deeply counterintuitive. Few people actually believe them. [...] Most people [...] believe what they see is, that some things are just plain True—and that they know what they are. Instead, we stress that human perception and knowledge are limited, that we operate from the basis of mental models, that we can never place our mental models on a solid foundation of Truth because a model is a simplification, an abstraction, a selection, because our models are inevitably incomplete, incorrect—wrong.[...] Yet we must recognize the inherent tension between being humble about the limitations of our knowledge on the one hand, and being able to argue for our views, respond to criticism, and make decisions on the other. Developing the capacity to see the world through multiple lenses and to respect differences cannot become an excuse for indecision, for a retreat to impotent scholasticism. We have to act. We must make the best decisions we can despite the inevitable limitations of our knowledge and models, then take personal responsibility for them.”[268]

¹ e.g. construction plans of Roddenberry’s *U.S.S. Enterprise NCC-1701* or maps of Tolkien’s *Middle-earth*

² proponent of System Dynamics, well known for his book “Business Dynamics: Systems Thinking and Modeling for a Complex World”[267]

The models presented in this thesis were chosen deliberately. Partially, others could have been selected or complement the presented ones. But within this thesis not all models and their inherent concepts can be presented. The selection is focused on those, which contributed to the introduced *Integrated Approach* (cf. chapter 4 on page 179) the most and help to comprehend the decisions made in its development.

2.2 LEARNING THEORIES

Suppes defines “A *theory* is a scientifically acceptable set of principles offered to explain a phenomenon. Theories provide frameworks for interpreting environmental observations and serve as bridges between research and education.”[272]

“It is the theory
which decides what
can be observed.”
—Albert Einstein

As Schunk emphasizes “Effective teaching requires that we determine the best theoretical perspectives for the types of learning we deal with and draw on the implications of those perspectives for teaching. When reinforced practice is important for learning, then teachers should schedule it. When learning problem-solving strategies is important, then we should study the implications of information processing theory.”[240]

Three major learning theories emerged in the 20th century and shaped the study of learning:

- *behaviorist learning theory*,
- *cognitivist learning theory*, and
- *constructivist learning theory*. [107]

These theories are summarized in table 2.1 on the following page and described in more detail in the following sections.

2.2.1 Behaviorist Learning Theory

Developed in the late 19th century—with scientific method still in its earliest days—and dominating the psychology of learning for the first half of the 20th century, *behaviorism* represented a radical leap forward in terms of human science, replacing metaphysics and divine as sole model of explanation. [107]

Behaviorist learning theory “held that the scientific study of psychology must restrict itself to the study of observable behaviors and the stimulus conditions that control them.”[58] With this strong empirical focus, considerations were limited to stimulus and response, where a given input stimulates an observable, measurable and repeatable response. Internal mental states, consciousness or phenomena as understanding, reasoning, and thinking were ignored, as they were

	Behaviorism	Cognitivism	Constructivism
mind is considered as	passive (largely irrelevant) black box	computer	closed informational system
knowledge gets	deposited	processed	constructed
knowledge is	a correct input-output relation	adequate internal cognitive process	ability to cope with a given situation
learning goals	correct answers, intended behavior	correct methods to find answers	mastering complex situations
pattern	stimulus and response, classical conditioning, operant conditioning,	problem solving	construction
teaching strategy	teach	observe and support	cooperate
teacher is	authority	tutor	coach, facilitator, enabler
feedback is	specified externally	modelled externally	modelled internally

Table 2.1: Summary of major learning theories

not observable, hence not scientifically evaluable and not adduced to explain the acquisition of behavior.

Learning is explained in terms of external events. *Conditioning* is a key term used in different manifestations of behavioral theory, e.g. *classic conditioning* (Pavlov), *contiguous conditioning* (Guthrie) and *operant conditioning* (Skinner).

In behaviorist theory, a learner associates stimulus with a response. This behavior might be shaped through *positive* and *negative reinforcement*, resulting in an increased probability that antecedent behavior is shown again. *Positive* or *negative punishment*, in contrast, decreases the probability of occurrence of repeated preceding shown behavior. In this context, *positive* indicates an application, and *negative* indicates an omitting of a stimulus.

“Behavioral theories seem best suited to explain simpler forms of learning that involve associations, such as multiplication facts, foreign language word meanings, and state capital cities.”[240] “Behavioral learning theory lent itself not only to instructional design based on very specific and discrete learning steps, but also to mechanization of the process through new forms of learning technologies [...] intended to encourage practice and reinforcement of specific tasks [...]”[107] Examples of those learning technologies include *testing machines*, *teaching machines*, *programmed instructions* and *computer-assisted instruction*. In order to design instruction, taxonomies of learning objectives were considered important. Section 2.3 on page 38 describes a revised version of such a taxonomy.

Insights gained by behaviorist researchers “[...] were largely based on interpretations of experiments with laboratory animals. The resultant perspectives were influential, but also issues of significant debate.”[107]

In response to criticisms arguing that “his claims exceeded his evidence and that he could not prove or demonstrate empirically that the responses were the result of a particular stimulus,” Skinner created “a set of highly controlled conditions in which a discriminating stimulus could be defined and linked to a specific and particular response.” This approach resulted in models with narrow validity, “testable only under very limited and limiting conditions.”[107] This again caused major criticism. Chomsky[55] claimed “If he accepts the broad definitions, characterizing any physical event impinging on the organism as a stimulus and any part of the organism’s behavior as a response, he must conclude that behavior has not been demonstrated to be lawful. In the present state of our knowledge, we must attribute an overwhelming influence on actual behavior to ill-defined factors of attention, set, volition, and caprice. If we accept the narrower definitions, then behavior is lawful by definition (if it consists of responses); but this fact is of limited significance, since most of what the animal does will simply not be considered behavior. Hence, the psychologist either must admit that behavior is not lawful (or that he cannot at present show that it is [...]), or must restrict his attention to those highly limited areas in which it is lawful [...].”

Bransford et al.[58] state “Over time, radical behaviorism [...] gave way to a more moderate form of behaviorism [...] that preserved the scientific rigor of using behavior as data, but also allowed hypotheses about internal ‘mental’ states when these became necessary to explain various phenomena.”

2.2.2 *Cognitivist Learning Theory*

“Cognitive psychology’s reaction against the inability of behaviorism to account for much human activity arose mainly from a concern that the link between a stimulus and a response was not straightforward, that there were mechanisms that intervened to reduce the predictability of a response to a given stimulus, and that stimulus–response accounts of complex behavior unique to humans, like the acquisition and use of language, were extremely convoluted and contrived.”[305] Another major problem perceived was, that behaviorism was not able to explain most social behaviors.

“Nonetheless, cognitivism did not reject behaviorist science altogether but shifted the emphasis from external behavior to a focus on the internal mental processes and to understanding how cognitive processes could promote effective learning.”[107]

Fields contributing to cognitivism theory were linguistics, neurology, and psychology. This new kind of thought was supported and influenced massively by the invention of the computer, manifesting in terms like *Mind as Computer* (MAC), “*human information processing*” and *Cognitive Information Processing* (CIP). The CIP model describes the mind as a system that processes information and employs components to encode, store, retrieve and transform information utilizing procedures. It was focussed on understanding how this processing is accomplished.

The *schema theory* in cognitivist learning theory holds “that learning is easier if new subject matter is compared to existing knowledge and is structured or representational.”[107] Schemata can be described as dynamic abstracting memory structures providing context. “A schema is a more abstract representation than a direct perceptual experience. [...] our knowledge of the world is constantly interpreting new experience and adapting to it. These processes, which Piaget (1968) has called ‘assimilation’ and ‘accommodation’, [...] interact dynamically in an attempt to achieve cognitive equilibrium without which the world would be a tangled blur of meaningless experiences. [...] Not only does a schema serve as a repository of experiences; it provides a context that affects how we interpret new experiences and even directs our attention to particular sources of experience and information.”[305]

As well as behaviorism, cognitivist theory was build on an *objectivist epistemology*. This “[...] epistemology holds that knowledge is fixed and finite, and ultimately, knowledge is truth. Knowledge is something that the teacher has mastered, and which students must now similarly master by replicating the knowledge of the teacher. The pedagogies emphasized ‘transmitting information’ by the teacher as a way to ‘acquire knowledge’ by the student, reflected in such didactic approaches as lectures or their mechanized versions [...]”[107] That is why cognitivist learning theory is focussed on instructional design. Some proponents and leaders of cognitivist learning theory were contributing to behaviorist theory before, which results in some blurring between both theories.

As such, Gagnè is well known for his *theory of instruction*. This theory is composed of a *taxonomy of learning outcomes*³, corresponding *specific conditions* to achieve these outcomes and *nine events of instruction* to facilitate learning processes:

1. *Gaining attention* (e.g. by change of stimulus)
2. *Informing learners of the objectives* (by telling them, what they will be able to do afterward)

³ This taxonomy shows some similarity to Bloom’s taxonomy (cf. section 2.3) and contains five categories: *verbal information*, *intellectual skills*, *cognitive strategies*, *attitudes* and *motor skills*.

3. *Stimulating recall of prior learning* (by assisting in recalling relevant information, ranging from simple reminding to some practice activity)
4. *Presenting the content* (by displaying learning content with distinctive features)
5. *Provide “learning guidance”* (e.g. by suggesting meaningful organization)
6. *Eliciting performance* (by letting learners perform activity indicating successful learning without any penalties for not yet being perfect and providing guidance for performance improvement)
7. *Providing feedback* (by giving informative feedback, assisting learners to detect and correct their misconceptions)
8. *Assessing performance* (by requiring additional learner performance and delivering feedback)
9. *Enhancing retention and transfer* (by providing varied practice and reviews, might be built in earlier to provide examples and context) [78]

Gagnè and Medsker state, “The nine events of instruction have been proven effective in innumerable real-world projects. They ensure completeness of each act of learning, because the learner is well prepared before processing new content, and because practice, feedback, and assessment guarantee that observable student performance meets the required standards. [...] Many training failures can be attributed to omission of some of the nine events of instruction. Perhaps the most common pitfall is to include no practice or insufficient practice [...]” [230] By providing instructional designers with very clear steps, this instructional theory has had significant influence and prominence in educational practice.[107]

Two prominent examples of (learning) technologies that had been emerging with cognitive learning theory are *Artificial Intelligence (AI)* and *Intelligent Tutoring System (ITS)*. [107]

Shute and Psotka summarize: “A student learns from an ITS primarily by solving problems—ones that are appropriately selected or tailor-made—that serve as good learning experiences for that student. The system starts by assessing what the student already knows, the *student model*. The system concurrently must consider what the student needs to know, the *curriculum* (also known as the *domain expert*). Finally, the system must decide what curriculum element (unit of instruction) ought to be instructed next, and how it shall be presented, the *tutor* (or inherent teaching strategy). From all of these considerations, the system selects, or generates, a problem, then either works

out a solution to the problem (via the domain expert), or retrieves a prepared solution. The ITS then compares its solution, in real-time, to the one the student has prepared and performs a diagnosis based on differences between the two. [...] Other kinds of systems may not even have a tutor/coach present. For example, the strength of microworlds (exploratory environments) resides in the underlying simulation and explicit interfaces in which students can freely conduct experiments and obtain results quickly and safely. [...] these systems can be intrinsically motivating, in terms of generating interesting complexities that keep students interested in continuing to explore, while giving them sufficient success to prevent frustration.” [254]

The ITS community has been facing some major discussion and growing discontent, as the term *intelligence* is associated with *awareness* and the term “*intelligent tutoring system*” might be misleading.[107] “Simply put, ITS may promise too much, deliver too little, and constitute too restrictive a construct.”[254] One resulting suggestion is not to attempt to build “omnipotent” tutors but to create a “collection of computerized tools.”[254]

2.2.3 Constructivism

“Discovery, like
surprise, favors the
well prepared mind.”
—Jerome Bruner
(1961)

“Constructivism refers both to a learning theory (how people learn) and to an epistemology of learning (what is the nature of knowledge).”[107]

“A key assumption of constructivism is that people are active learners and develop knowledge for themselves [...] To understand [...], learners must discover the basic principles [...]. Constructivists differ in the extent to which they ascribe this function entirely to learners. Some believe that mental structures come to reflect reality, whereas others (radical constructivists) believe that the individual’s mental world is the only reality. Constructivists also differ in how much they ascribe the construction of knowledge to social interactions [...]”[240]

With a wide diversity of views, making the term *constructivism* serve as an umbrella term, proponents “seem to be committed to the general view that (1) learning is an active process of constructing rather than acquiring knowledge, and (2) instruction is a process of supporting that construction rather than communicating knowledge.”[65] Bodner[26], citing von Glasersfeld, states “[...] learners construct understanding. They do not simply mirror and reflect what they are told or what they read. Learners look for meaning and will try to find regularity and order in the events of the world even in the absence of full or complete information.”

Two of the most well-known representatives of constructivism are *Piaget* and *Vygotsky*, both emphasizing different aspects and taking partially contradicting positions. Piaget promoted *cognitive construc-*

tivism, concerned with “how the individual learner understands the world through biological stages,” while Vygotsky emphasized *social constructivism*, focused on “how meanings and understandings grow out of social encounters.”[107]

Piaget, primarily interested in epistemology, had been taken the view that “looking carefully at how knowledge develops in children will elucidate the nature of knowledge in general.” In his belief “children are not empty vessels to be filled with knowledge [...] but active builders of knowledge, little scientists who are constantly creating and testing their own theories of the world.”[285] “Piaget believed that knowledge is acquired as the result of a life-long constructive process in which we try to organize, structure, and restructure our experiences in light of existing schemes of thought, and thereby gradually modify and expand these schemes.”[26]

“Jean Piaget’s theories about children and learning include the concepts of *assimilation* and *accommodation*. With assimilation, we attempt to fit new information into existing slots or categories. [...] Accommodation involves the process whereby we must modify our existing model of the world to accommodate new information that does not fit into an existing slot or category. This process is the result of holding two contradictory beliefs. [...] This process is often referred to as *cognitive disequilibrium*. [...] Piaget believed that intellectual maturation over the lifespan of the individual depends on the cycle of assimilation and accommodation and that cognitive disequilibrium is the key to this process.”[295, emphasis added]

Vygotsky argued that “social interactions are an essential part of human cognitive development” and that “biological development does not occur in isolation”[107]. Social and cultural influences shape our thoughts and language. According to his concept of *Zone of Proximal Development* (ZPD) “learning takes place when learners solve problems beyond their actual developmental level—but within their level of potential development—under [...] guidance or in collaboration with [...]” a *More Knowledgeable Other* (MKO)⁴, who “[...] supports the learner by providing the tools (language, concepts) needed to advance and eventually independently achieve the learner’s intended goal.”[107]

“Constructivism resonated with practicing teachers and became a highly popular concept in the field. However, neither Piaget nor Vygotsky had ever written about the implications of their theories for the classroom, and hence the resulting constructivist pedagogies and technologies were primarily attempts by practitioners to implement notions of active learning.”[107]

“A common misconception regarding ‘constructivist’ theories of knowing (that existing knowledge is used to build new knowledge)

⁴ A skilled partner, someone who has a better understanding or a higher ability level than the learner, with respect to a particular task, process, or concept. This might be an advanced peer, teacher, parent, etc.

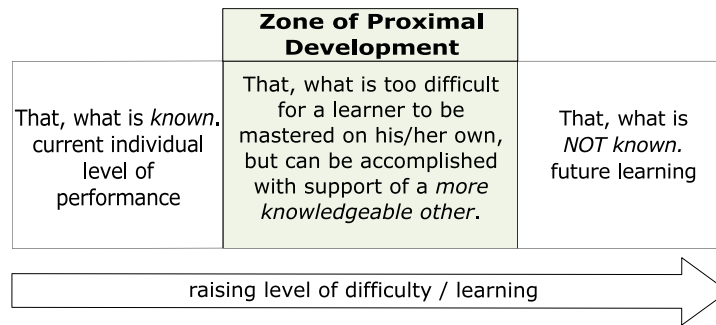


Figure 2.3: Zone of Proximal Development (Vygotsky)

is that teachers should never tell students anything directly but, instead, should always allow them to construct knowledge for themselves. This perspective confuses a theory of pedagogy (teaching) with a theory of knowing. Constructivists assume that all knowledge is constructed from previous knowledge, irrespective of how one is taught [...] even listening to a lecture involves active attempts to construct new knowledge.”[58]

Bok claims “By some calculations, the average student will be unable to recall most of the factual content of a typical lecture within fifteen minutes after the end of class. In contrast, interests, values, and cognitive skills are all likely to last longer, as are concepts and knowledge that students have acquired not by passively reading or listening to lectures but through their own mental efforts.”[33]

A number of teaching approaches, or pedagogies⁵, and principles were created based on the constructivist learning theory in order to facilitate *own mental efforts* of students.

Active Learning and *Learning by Doing* are learner-centered approaches that let learners engage in an activity instead of just hear or read about it. Activities can be manifold. A teacher, or MKO in general, encourages and assists in constructing knowledge of a subject, e.g. by providing useful resources. Discussions while and after engaging in the activity foster reflection. The activity should be built around an authentic and complex enough real-world problem that is inviting the learner to apply existing knowledge, hypothesize, test hypotheses and draw conclusions.[107, 218]

While engaging in an activity, it is of great value to be allowed to be *learning from failures*. Proponents, in fact, recommend to set up learners to fail in order to increase learning outcome.[218, 236].

⁵ Defined by the Oxford Dictionary of English as: “pedagogy, noun (pl. pedagogies), the method and practice of teaching, especially as an academic subject or theoretical concept [...]”[72]

Anchored Instruction “is designed to help students develop useful knowledge rather than inert knowledge.” It emphasizes “the importance of creating an anchor of focus that generates interest and enables students to identify and define problems and to pay attention to their own perception and comprehension of these problems. They can then be introduced to information that is relevant to their anchored perceptions. The major goal [...] is to experience the changes in their perception and understanding of the anchor as they view the situation from new points of view.”[39] While the creators of this approach favored videos, as those were state of the art at that time, anchors can be manifold.

Bruner, who coined the term *Discovery Learning*, states “Our aim as teachers is to give our student as firm a grasp of a subject as we can, and to make him as autonomous and self-propelled a thinker as we can—one who will go alone on his own after formal schooling has ended.”[46] To enable students in such ways, they should be empowered to benefit “from the experience of learning through discoveries that one makes for oneself.” According to Bruner, this approach results in *increased intellectual potency, a shift from extrinsic to intrinsic rewards, learning the heuristics of discovering, and aid to memory processing*. [46]

But Alfieri et al. claim “however, there is a myriad of discovery-based learning approaches presented within the literature without a precise definition. [...] Learning tasks considered to be within the realm of discovery learning range from implicit pattern detection to the elicitation of explanations, and from working through manuals to conducting simulations. What exactly constitutes a discovery-learning situation is seemingly yet undetermined by the field as a whole. [...] Common to all of the literature, however, is that the target information must be discovered by the learner within the confines of the task and its material.”[10]

Bruner stated, “it goes without saying that, left to himself, the child will go about discovering things for himself within limits.”[46] As he claimed, “discovery, like surprise, favors the well prepared mind.”[46] This preparation not merely include knowledge about the domain but experience with the act of discovery itself too. Recent research supports this statement, claiming that “such methods as *extreme* modes of discovery [...] with almost no teacher guidance will, of course, be inferior to more guided methods.”[10] Alfieri et al. suggest “enhanced-discovery tasks requiring learners to be actively engaged and constructive seem optimal. On the basis of the current analyses, optimal approaches should include at least one of the following: (a) guided tasks that have scaffolding in place to assist learners, (b) tasks requiring learners to explain their own ideas and ensuring that these ideas are accurate by providing timely feedback, or (c) tasks that pro-

vide worked examples of how to succeed in the task.”[10]

In *Problem-Based Learning* students, working in groups, are faced with a problem that has no correct solution. Learners, assisted by facilitators, have to identify what they need to know to solve the problem. This “requires students to think creatively and bring their knowledge to bear in unique ways. It is especially useful for projects that have no one correct solution. [...] Students who collaborate to solve problems become aware of new ways that knowledge can be used and combined, which forms new synaptic connections. Further, problem-based learning is apt to appeal to students’ motivation and engender emotional involvement, which also can create more extensive neural networks.”[240]

Characteristics of *Situated Learning* are described by Brown et al.[45] “[...] people who use tools in authentic activity actively build an increasingly rich implicit understanding both of the tool themselves and of the world in which they use those tools. Their understanding, initially narrow, is continually broadened through use. Learning and acting are, as a result, interestingly indistinct, learning being a continuous, life-long process resulting from acting in situations.” As a consequence of these findings the environment, in which learners engage, should closely resemble the real-life environment, where this knowledge will be applied.

Scaffolded Learning is strongly connected to Vygotsky’s concept of the *zone of proximal development* (ZPD). “Scaffolding gives students a context, motivation, and foundation from which to understand the new information. In order for learning to progress, scaffolds should be gradually removed as the learner progresses, so that students will eventually be able to demonstrate comprehension independently.”[107]

Collaborative Learning is not an educational approach on its own but might be utilized with any of the mentioned approaches. As emphasized by Vygotsky’s *social constructivism* learners benefit from collaboration. This collaboration should invite to share alternative perspectives on a subject, articulating and discussing them—or developing new ones. It should promote peer tutoring, dialogical interchange and reflexivity on the learning subject and the learning process.[65]

Jonassen [134] proposed several characteristics that facilitate learning in constructivist learning environments. Such environments should:

1. provide multiple representations of reality to avoid oversimplification,
2. represent the complexity of the real world,

3. emphasize knowledge construction, not knowledge reproduction,
4. emphasize authentic tasks in a meaningful context rather than abstract instruction out of context,
5. provide learning environments such as real-world settings or case-based learning instead of predetermined sequences of instruction,
6. encourage thoughtful reflection on experience,
7. enable context- and content-dependent knowledge construction, and
8. support collaborative construction of knowledge through social negotiation, not competition among learners for recognition.

“Jonassen’s list has been accepted by both social and cognitivist constructivists, albeit with some differences in emphasis.”[107]

Constructivist teaching approaches appear promising but are not without criticism. “Constructivist pedagogies have developed outside the learning theories developed by Piaget or Vygotsky. [...] The role of the teacher has been unsettled [...] without clear alternatives.”[107] A common slogan describing the role of the constructivist teacher is “*guide on the side*” not “*sage on the stage*”, emphasizing a role as facilitator. In reality, a combination of both teaching approaches might be necessary. A teacher has to select the most appropriate approach for her students and the content, favoring constructivist approaches when it comes to content that requires problem-solving and critical thinking skills.

Another criticism is concerned with too contextualized learning. “Simply learning to perform procedures, and learning in only a single context, does not promote flexible transfer. The transfer literature suggests that the most effective transfer may come from a balance of specific examples and general principles, not from either one alone.”[58]

Critics of minimally guided instruction state: “when students learn science in classrooms with pure-discovery methods and minimal feedback, they often become lost and frustrated, and their confusion can lead to misconceptions.”[144] They argue, “there is a growing body of research showing that students learn more deeply from strongly guided learning than from discovery,” and argue that “cognitive load theory suggests that the free exploration of a highly complex environment may generate a heavy working memory load that is detrimental to learning. This suggestion is particularly important in the case of

novice learners, who lack proper schemas to integrate the new information with their prior knowledge.”[144] It is suggested that “guidance can be relaxed only with increased expertise as knowledge in long-term memory can take over from external guidance.” Actually, this is the core of the concepts of *zone of proximal development* and *Scaffolded Learning*, both described above. Guidance and collaboration with a *more knowledgeable other*, scaffolds respectively, have to be designed carefully to provide the support needed by the learner.

In response to those criticisms[144] proponents of *Problem-Based Learning (PBL)* that was explicitly criticized argue that PBL “cannot be equated with minimally guided instruction. On the contrary, we contend that the elements of PBL allow for flexible adaptation of guidance, making this instructional approach potentially more compatible with the manner in which our cognitive structures are organized than the direct guided instructional approach [...]”[238] They describe elements of PBL, show how they are being used in contemporary curricula, and “present the multiple ways in which intrinsic, extraneous, and germane cognitive load can be managed through these elements.”[238]

2.2.4 *Beyond Constructivism*

Learning theories are an ongoing dynamic field of research and did not stop with *Constructivism*. “Today, the world is in the midst of an extraordinary outpouring of scientific work on the mind and brain, on the processes of thinking and learning, on the neural processes that occur during thought and learning, and on the development of competence.”[58]

Harasim[107] notes: “The 21st century is referred to as the Knowledge Age, a time in which knowledge has key social and economic value.” But as Bransford et al.[58] state “above all, information and knowledge are growing at a far more rapid rate than ever before in the history of humankind. As Nobel laureate Herbert Simon wisely stated, the meaning of ‘knowing’ has shifted from being able to remember and repeat information to being able to find and use it [...] the sheer magnitude of human knowledge renders its coverage by education an impossibility; rather, the goal of education is better conceived as helping students develop the intellectual tools and learning strategies needed to acquire the knowledge that allows people to think productively.”

Siemens[256] agrees “Our ability to learn what we need for tomorrow is more important than what we know today. A real challenge for any learning theory is to actuate known knowledge at the point of application. When knowledge, however, is needed, but not known, the ability to plug into sources to meet the requirements becomes a vital skill. As knowledge continues to grow and evolve, access to

what is needed is more important than what the learner currently possesses.”

He[256] proposes the *Connectivism* learning theory that is providing “insight into learning skills and tasks needed for learners to flourish in a digital era.” Among others, Siemens[256] states that “technology is altering (rewiring) out brains. The tools we use define and shape our thinking. [...] Know-how and know-what are being supplemented with know-where (the understanding of where to find knowledge needed).”

Siemens[256] describes his theory as “the integration of principles explored by chaos, network, and complexity and self-organization theories. Learning is a process that occurs within nebulous environments of shifting core elements—not entirely under the control of the individual. Learning (defined as actionable knowledge) can reside outside of ourselves (within an organization or a database), is focused on connecting specialized information sets, and the connections that enable us to learn more are more important than our current state of knowing.”

A *Massive Open Online Course (MOOC)* provides an example of applied *Connectivism*. Participants in a MOOC are guided by facilitators but are largely responsible for what they learn and how they share their new knowledge and hence create parts of the course content.

With her *Online Collaborative Learning (OCL) Theory* Harasim[107] addresses perspectives somewhat similar to *Connectivism*.

However with new perspectives, theories, or new variants of existing ones emerging, the field seems yet to be rather unclear, much in flow and discussion as well as not yet widely accepted.

2.2.5 Two Instructional Design Theories

This section describes two instructional design theories that were chosen because they are, besides constructivist learning approaches, providing specific guidance and valuable insights for the design of the *Integrated Approach* introduced in chapter 4. With roots in cognitivist learning approaches, both theories have been continuously developed since their initial introduction. While the proponents of the first one, *Cognitive Load Theory (CLT)*, argue rather against constructivist educational approaches, actually against minimally or unguided versions of it, the second one, *Elaboration Theory*, was explicitly updated to include such approaches.

2.2.5.1 Sweller’s Cognitive Load Theory (CLT)

Based on a model of human cognitive architecture, with foundations in evolutionary principles, *Cognitive Load Theory (CLT)* is concerned with identifying aspects of human cognition relevant to instructional

<i>biologically primary knowledge</i>	<i>biologically secondary knowledge</i>
knowledge we have evolved to acquire	cultural knowledge we have not evolved to acquire
modular, with different types of knowledge unrelated to each other and acquired independently at different times and in different ways	types of knowledge that bear some relation to each other and are acquired in similar manner
acquired easily, automatically and unconsciously	acquired deliberately with conscious effort
explicit instruction not required	best acquired with explicit instructions
e.g. imitating others, listening to and speaking our native language, means-ends analysis, random generate and test procedures	e.g. reading, listen to and speaking non-native language, following complex defined methods or processes

Table 2.2: Cognitive Load Theory: Distinctions Between biologically primary and secondary knowledge [195, examples added]

issues. This theory has been developed and refined over several decades and tested resulting hypotheses using randomized controlled experiments. Proponents argue that “without knowledge of relevant aspects of human cognitive architecture such as the characteristics and intricate relations between working memory and long-term memory, the effectiveness of instructional design is likely to be random.”[194]

Dividing knowledge into two, evolutionary based, categories⁶, *biologically primary knowledge* and *biologically secondary knowledge*, CLT is focused primarily on the latter category, whose acquisition is leveraged by the first category. Table 2.2 summarizes characteristics of both knowledge categories.

The acquisition of the secondary category is determined by human cognitive architecture that is an example of a natural information processing system. Such a system requires a very large store of information, which is provided by the long-term memory in human cognition. This one “heavily determines our cognitive lives,” because “almost all human cognitive activity is determined by information held in long-term memory.”[195]

CLT defines learning as “[...] alteration in long term memory. If nothing has altered in long-term memory nothing has been learned. Accordingly, appropriate alteration of long term memory’s store of biologically secondary information should be the primary aim of instruction.”[195] According to CLT both, *rote learning* and *understanding*, underlie the same principle of change in long-term memory, but *understanding* is enabled by additional changes in long-term memory providing essential connections between elements that are omitted in rote learning.

⁶ borrowed from Geary[92]

Problem-solving skill is critically determined by information in long-term memory. "Experts have a vastly superior memory to novices for problem states in their field of expertise. [...] Such knowledge [...] allows an expert to immediately recognize most of the situations faced and the actions required by that situation. [...] it] permits the fluency shown by experts in their own area." [195] A learner, without external guidance, facing a new problem has to engage in problem-solving to find a way. "If information is not available, the student must discover the new procedures required using a random generate and test procedure." [195]

CLT states that capacity and duration of working memory are severely constrained. "When dealing with novel, biological secondary information, human working memory has two severe limitations. [...] working memory is able to hold only about 7 elements of information. It can probably process [...] no more than about 2-4 novel elements. [...] without rehearsal, almost all the contents of working memory are lost within about 20 seconds." [195] Working memory determines, which information should be used to alter the information store in long-term memory. CLT holds that these limits might be protective to ensure functionality of the long-term memory information store.

According to CLT those limits do not exist for information, fed from long-term memory into working memory, neither in capacity nor duration. In that way, information from long-term memory vastly extends capability of working memory. The relation between both types of memory is described by Paas and Sweller in the following manner. "At one end of the continuum, when one is dealing with unfamiliar information, working memory limitations are critical. They become successively less critical as familiarity increases, that is, as more and more information from long-term memory is used. At the other extreme, when one is dealing with information incorporated in well-entrenched knowledge, working memory limitations become irrelevant. Thus, the extent to which working memory limitations matter depends on the extent to which the information being dealt with has been organized in long-term memory." [195]

CLT emphasizes that understanding is bound to changes in long-term memory. "Without changes in long-term memory, nothing has been understood. [...] Understanding occurs when all relevant elements of [...] information can be processed simultaneously in working memory. [...] there may be too many elements to simultaneously process in working memory. If the elements are essential, understanding cannot occur until it becomes possible to process them. While the learner is studying the material, elements are organized and combined into knowledge held in long-term memory. When knowledge acquisition has progressed to the point where all of the elements essential to understanding a topic can be processed in working memory, understanding has occurred. [...] understanding can be defined

Category	Source	Implications
Intrinsic	caused by natural complexity, interacting elements that are intrinsic to the task and must be processed simultaneously,	cannot be altered other than by changing the nature of the task or by increasing knowledge level of learner; maybe combine multiple elements into single element during learning
Extraneous	caused by interacting elements introduced by an, potentially inappropriate, instructional design; require learner to use working memory resources to process elements that do not lead to knowledge acquisition	should be reduced by altering instructional design, e.g. by removing unnecessarily introduced large number of interacting elements
Germane	'effective' cognitive load; refers to working memory resources dealing with intrinsic rather than extraneous cognitive load, thus facilitating learning	the higher the value, the more effective will be instruction

Table 2.3: Cognitive Load Theory: The 3 Categories of Cognitive Load [adapted from 195]

as the ability to simultaneously process required elements in working memory.”[195]

CLT distinguishes three kinds of cognitive load:

- *intrinsic*,
- *extraneous*, and
- *germane* cognitive load.

Table 2.3 summarizes characteristics and implications of these categories.

Both extraneous and intrinsic cognitive load have to be managed by the constrained working memory. They are additive. Essentially the extraneous load, introduced by the chosen instructive design, should be kept to the minimum necessary to acquisition of the intended knowledge.

Rooted in cognitivist learning theory, proponents of CLT criticize minimally or unguided instruction methods. They argue that such instructional designs entail unnecessary extraneous cognitive load interfering learners' knowledge acquisition. Instead, they recommend the use of worked examples provided for comprehension.[144, 273, 274]

2.2.5.2 Reigeluth's Elaboration Theory

The *Elaboration Theory* [224, 225] states “[...] that if cognitive instruction is organized in a certain specified way, then that instruction will result in higher levels of learning, synthesis, retention, and affect.”[224]

This approach is focused on when information is presented to the learner and how it should be scoped and sequenced. It holds that instruction should be organized from *general to detailed*, from *simple to complex* and from *abstract to concrete*.

Simplest versions of a task should be introduced *before more complex* versions.

“The general-to-detailed organization prescribed by the elaboration model helps to ensure that the learner is always aware of the context and importance of the different topics that are being taught. It allows the learner to learn at the level of detail that is most appropriate and meaningful to him or her at any given state in the development of one’s knowledge.”[224]

The second revision of this theory adds guidance on scoping learning content and introduces a new set of proposed sequencing types for new constructivist educational approaches: “most of the new approaches to instruction, including simulations, [...] goal-based scenarios, problem-based learning, and other types of situated learning, require a more holistic approach to sequencing, one that can simplify the content or task, not by breaking it into pieces, but by identifying simpler real-world versions of the task or content domain.”[225]

The *Elaboration Theory* provides different types of methods to sequence learning content corresponding to the type of learning objectives but sharing a number of characteristics. This theory holds that broader more inclusive concepts or principles should be provided before narrower, more detailed concepts that elaborate upon them, that “*supporting*” content, e.g. related concepts, principles, procedures, higher-order thinking skills, attitudes, etc., should be provided together with the concepts or principles, which they are most closely related to.[225] Learning content, together with its supporting content, should be grouped into “[...] ‘learning episodes’, that aren’t so large as to make review and synthesis difficult but aren’t so small as to break up the flow of the learning process.”[225]

Reigeluth[225] states with regards to complex tasks (with a focus on skills) that “the simplifying conditions method [...] sequencing strategy enables learners to understand the tasks holistically and to acquire the skills of an expert for a real-world task from the very first [...] ‘learning episode’ [...]. These skills enhance the motivation of learners and, therefore, enhance the quality (effectiveness and efficiency) of the instruction. The holistic understanding of a task results in the formation of a stable cognitive schema to which more complex capabilities and understandings can be assimilated. [...] since the learners start with a real version of the task from the beginning, this method is ideally suited to situated learning, problem-based learning, computer-based simulations [...] Further, it can be used with highly directive instruction, highly constructivist instruction, or anything in between.”

2.2.6 Conclusions

This section introduced a selection of learning theories as well as supporting theories and concepts, that are of interest for the *Integrated Approach* introduced in this thesis.

Learning theories evolved over time, based on and in reaction to preceding theories. They reflect in large part social and technological developments and achievements of their time and shifted the view on knowledge and learning.

Driven by passionate proponents and being influenced by many fields of science and research it is only natural that none of these theories, teaching approaches, pedagogies or instructional designs is without criticisms.

We find more radical proponents of their theories and more moderate and pragmatic ones. With a sound knowledge of theories and concepts, it is up to the instructional designer to choose and combine the most appropriate ones to optimize learning experiences.

As Schunk emphasizes “Effective teaching requires that we determine the best theoretical perspectives for the types of learning we deal with and draw on the implications of those perspectives for teaching. When reinforced practice is important for learning, then teachers should schedule it. When learning problem-solving strategies is important, then we should study the implications of information processing theory.”[240]

In the SE education related context, a number of studies argue for constructivist learning approaches, e.g. in engineering education [131, 170, 259], distance learning and distributed learning environments[68, 102, 275], computer science education [20, 21, 60] and SE education itself [94, 103]. All game-based approaches presented in section 2.5 on page 68 argue for constructivist approaches.

When it comes to learning just definitions, word meanings, behaviorist instructional approaches may serve well, but for the objectives of this research, it will by far not be enough. Definitions of terms may be a starting point. But to grasp concepts, (inter-)relationships of elements as well as their dynamic behavior have to be understood. As stated in section 1.4 on page 9, primary objectives of this research work include to provide students in SE courses with *competencies*⁷, to keep a holistic view on the development endeavor—in all its phases and with all its relevant dimensions—while working focused on details required by it. These objectives require mastering complex situations and the ability to cope with them, once they appear. According to the remarks given in preceding sections, this calls for *skills-oriented*,

⁷ Cf. section 2.3.2 on page 40 for the definition of *competency* utilized in this thesis.

active, problem-based, situated, and collaborative constructivist learning approaches (cf. section 2.2.3 and table 2.1). Jonassen's[134] list (cf. section 2.2.3 on page 28) provides important widely accepted characteristics of such learning environments. As proposed by *Anchored Instruction*, a collective learning experience may provide an *anchor* used in subsequent learning activities.

As stated in the preceding sections, constructivist instructional approaches were criticized in several ways, most notably for minimally guided approaches and too contextualized situated learning.

Following a *scaffolded* learning approach, providing the necessary guidance may address issues connected to too less guidance. Instructional approaches rooted in cognitivist learning theory may support to provide scaffolding in the right way and to structure the learning experiences provided to students.

Following complex methods or processes represents *biologically secondary knowledge* from the perspective of *Cognitive Load Theory* (CLT, cf. table 2.2) and requires learner's conscious effort and guidance to be deliberately acquired. CLT reminds us that working memory of learners is limited and provides arguments to focus on the right type of cognitive load as well as arguments to omit extraneous cognitive load. From the perspective of CLT, each concept learned and already organized in long-term memory supports at coping with new related situations and learning content.

Elaboration Theory provides concepts to structure, scope, and sequence learning content to enable learners to understand tasks holistically. As Reigeluth[225] states, providing a learning task with simplified conditions first "enables learners to understand the tasks holistically and to acquire the skills of an expert for a real-world task from the very first [...] 'learning episode' [...]. These skills enhance the motivation of learners[...]. The holistic understanding of a task results in the formation of a stable cognitive schema to which more complex capabilities and understandings can be assimilated. [...] this method is ideally suited to situated learning, problem-based learning, computer-based simulations [...]."

Both, CLT as well as Elaboration Theory, are widely accepted, have been proving their utility in education over decades and underwent continuous advancement since their introduction decades ago.

Taking all this into account, a successful approach providing the intended learning objectives should combine *social* as well as *cognitive constructivist learning* approaches with the guidance provided by *cognitivist instructional approaches*. For that, the *Cognitive Load Theory* (CLT) and *Elaboration Theory* were chosen for the reasons presented in the sections above.

2.3 LEARNING OBJECTIVES AND COMPETENCIES

To successfully design a learning activity, course unit, course, or curriculum, it is of particular importance to have a clear understanding of the learning objectives, knowledge, skills, attitudes, or competencies⁸, at which the particular activity is aiming.

For the purpose of this research work, it is important to identify competencies, skills, knowledge and attitudes that are expected to be of high interest for SE education to find evidence for its relevance.

On the other hand, it is important to locate the approaches, proposed in this thesis, inside of curriculum guidelines provided by relevant organizations.

But before the field of competencies and curriculum guidelines gets entered, a widely used and accepted model of learning objectives gets introduced.

2.3.1 *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*

2.3.1.1 *The Original Taxonomy*

Originating in 1956, the *Taxonomy of Educational Objectives*[25], known as *Bloom's Taxonomy*, is a framework to classify expected or intended learning objectives as result of instruction. Initially, it was developed to facilitate the exchange of test items among universities to reduce the annual effort of preparing comprehensive examinations.[147] Additional to that goal, it was created to serve as common language and means of educational objectives to compare educational activities, courses, or curricula.

The original taxonomy provided six major categories in the cognitive domain: *Knowledge, Comprehension, Application, Analysis, Synthesis, and Evaluation* (ordered from simple to complex). All but *Application* were provided with subcategories. "Further, it was assumed that the original Taxonomy represented a cumulative hierarchy; that is, mastery of each simpler category was prerequisite to mastery of the next more complex one." [12]

To define an intended learning objective, a subject matter content (a noun) and a description of what to be done with or to it (a verb representing the cognitive process) have to be combined, usually prefixed with a statement like "the learner will be able to." To give an example: "The student will be able to remember the phases of the OpenUP." In

⁸ The Oxford Dictionary of English defines *competency* as "The ability to do something successfully or efficiently." [72] We will see in section 2.3.2 on page 40 that this definition is specified more precisely (in multiple ways) by players in the competencies field.

this example the subject matter content (the noun) is “phases of the OpenUP,” and the cognitive process (the verb) is “remember.” This learning objective could be categorized with the Bloom’s Taxonomy level *Knowledge*⁹. Sets of verbs characterizing specific levels were collected to simplify the categorization of learning objectives.

As “readers saw its potential, the framework became widely known and cited, eventually being translated into 22 languages.”[12] The taxonomy was frequently used to classify and analyze intended curricular learning outcomes, e.g. in SE education[132, 133, 269] and training[38]. “Such analyses [...] have repeatedly provided a basis for moving curricula and tests toward objectives that would be classified in the more complex categories.”[12]

2.3.1.2 The Revised Taxonomy

To reflect the increased understanding of cognitive learning that had emerged over decades after publishing the original version of the taxonomy, and to take into account feedback and empirical research regarding the taxonomy, a revised version was edited and published in 2001[12].

Instead of the original one-dimensional taxonomy, the revised version provides a two-dimensional framework.

The first dimension *Knowledge* resembles the subcategories of the original *Knowledge* category. Table 2.4 on page 41 summarizes this dimension. The last category *Metacognitive Knowledge* was added to reflect the “importance of students being made aware of their metacognitive activity, and then using this knowledge to appropriately adapt the ways in which they think and operate.”[147]

The second dimension *Cognitive Process* retained the original number of categories but underwent important changes. The original category *Knowledge* was renamed *Remember*. In reaction to frequent criticism the *Comprehension* category was renamed *Understand*. The *Synthesis* category was renamed *Create* and changed places with *Evaluation*, now called *Evaluate* to consistently use verb forms. Table 2.5 on page 42 summarizes the elements—representing cognitive processes—of the *Cognitive Process Dimension*. This dimension contains six categories that again include 19 subcategories.

“Like the original Taxonomy, the revision is a hierarchy in the sense that the six major categories of the Cognitive Process dimension are believed to differ in their complexity, with remember being less complex than understand, which is less complex than apply, and so on. However, because the revision gives much greater weight to teacher usage, the requirement of a strict hierarchy has been relaxed to allow

⁹ particularly 1.11 *Knowledge of terminology*, but the usage of only the top level category found wider utilization

the categories to overlap one another.”[147]

Having two dimensions to describe an expected learning objective, they can be described and easily visualized using a matrix, where both dimensions form the axes of the table. Table 2.6 shows such a table. By categorizing learning objectives of an activity, unit, or course and subsequently entering them into the *Taxonomy Table*, the objectives get clearly visualized. “In addition to showing what was included, the Taxonomy Table also suggests what might have been but wasn’t.”[147] This might trigger further thinking about the intended learning outcomes in order to optimize them.

“Problem solving and critical thinking were two other terms commonly used by teachers that were also considered for inclusion in the revision. But unlike understand, there seemed to be no popular usage that could be matched to a single category. Therefore, to be categorized in the Taxonomy, one must determine the intended specific meaning of problem solving and critical thinking from the context in which they are being used.”[147]

Both, the original version as well as the revised version and adapted variants, are utilized to define and describe intended learning outcomes of SE curricula and courses.[43, 81, 95, 132, 133, 169, 269, 283, 284]

2.3.2 Demanded Software Engineering (SE) Competencies

Although the term competency is “increasingly used”[270] and “one of the most popular terms since the beginning of the 21st century and the introduction of the Bologna process”[243], there is still no widely accepted definition.[270] Constructivist proponents[270] argue “The one and only true competence definition does not exist, nor will ever be found. [...] Here, the criterion for a competence definition is not whether the definition is true but the extent to which the constructed definition has proved to be adequate in the context in which it is used.”


Along these lines the following definition, based and drawing on existing definitions [14, 81, 95, 283], is proposed in the context of this work/thesis:

Competency—A cluster of related *knowledge*, *skills*, and *attitudes* enabling a person to accomplish all the tasks in a given context, that correlates with measurable performance, which can be improved by education, training and experience.

Knowledge in this definition represents, *what one knows*.

<p>A. Factual Knowledge—<i>The basic elements that students must know to be acquainted with a discipline or solve problems in it.</i></p> <p>Aa. Knowledge of terminology</p> <p>Ab. Knowledge of specific details and elements</p>	concrete knowledge
<p>B. Conceptual Knowledge—<i>The interrelationships among the basic elements within a larger structure that enable them to function together.</i></p> <p>Ba. Knowledge of classifications and categories</p> <p>Bb. Knowledge of principles and generalizations</p> <p>Bc. Knowledge of theories, models, and structures</p>	
<p>C. Procedural Knowledge—<i>How to do something; methods of inquiry, and criteria for using skills, algorithms, techniques, and methods.</i></p> <p>Ca. Knowledge of subject-specific skills and algorithms</p> <p>Cb. Knowledge of subject-specific techniques and methods</p> <p>Cc. Knowledge of criteria for determining when to use appropriate procedures</p>	
<p>D. Metacognitive Knowledge—<i>Knowledge of cognition in general as well as awareness and knowledge of one's own cognition.</i></p> <p>Da. Strategic knowledge</p> <p>Db. Knowledge about cognitive tasks, including appropriate contextual and conditional knowledge</p> <p>Dc. Self-knowledge</p>	abstract knowledge

Table 2.4: Revised Taxonomy: Knowledge Dimension[12]

<p>1 Remember—Retrieving relevant knowledge from long-term memory.</p> <p>1.1 Recognizing</p> <p>1.2 Recalling</p>	<p>lower order thinking skills</p> 
<p>2 Understand—Determining the meaning of instructional messages, including oral, written, and graphic communication.</p> <p>2.1 Interpreting</p> <p>2.2 Exemplifying</p> <p>2.3 Classifying</p> <p>2.4 Summarizing</p> <p>2.5 Inferring</p> <p>2.6 Comparing</p> <p>2.7 Explaining</p>	
<p>3 Apply—Carrying out or using a procedure in a given situation.</p> <p>3.1 Executing</p> <p>3.2 Implementing</p>	
<p>4 Analyze—Breaking material into its constituent parts and detecting how the parts relate to one another and to an overall structure or purpose.</p> <p>4.1 Differentiating</p> <p>4.2 Organizing</p> <p>4.3 Attributing</p>	
<p>5 Evaluate—Making judgments based on criteria and standards.</p> <p>5.1 Checking</p> <p>5.2 Critiquing</p>	
<p>6 Create—Putting elements together to form a novel, coherent whole or make an original product.</p> <p>6.1 Generating</p> <p>6.2 Planning</p> <p>6.3 Producing</p>	

higher
order
thinking
skills

Table 2.5: Revised Taxonomy: Cognitive Process Dimension[12]

		The Cognitive Process Dimension					
		Remember	Understand	Apply	Analyze	Evaluate	Create
The Knowledge Dimension	Factual						
	Conceptual						
	Procedural						
	Metacognitive						

Table 2.6: Taxonomy Table of The Revised Taxonomy[12]

Skills in this definition represent, *what one can do*. This includes *capabilities* and *abilities* and involves *knowledge*.

Attitudes in this definition are concerned with the ability to apply knowledge and skills in an effective manner, characterized by attributes of behavior such as *willingness, initiative, communicativeness, cooperativeness, self-reflection, trustworthiness, empathy, cultural and social sensitivity, professionalism, problem awareness, creativity* (if needed), etc.

The body of literature about competencies, their definition, their categorization, their grading, their assessment and their implications is ever-growing. Driven by varying motivations, competency models with different purposes, different definitions, categorizations, and emphasis were developed.

The number of contributions to conferences shows that competencies in SE education have been of particular interest to researchers, educators and curriculum planners[36, 242, 284] recently. Subjects of research include the competencies needed for a defined group of learners[97, 169], how to identify them[242, 284, 288], how to define them [235, 284, 289], how to categorize them[43, 264], how to measure or assess them[271], and how well they fit to various educational approaches[36, 241, 262].

National[48, 235] and international research programs[277] to examine the field of competencies were conducted. So it comes to no surprise that almost any bigger organization has own competency models tailored to their special needs, which include [6, 14, 81, 283].

For this thesis, developments in the field of SE and related disciplines are of interest. So it is primarily focused on those.

The SWEBOK in its 2004 version[4] contains an “Appendix D - Classification of Topics According to Bloom’s Taxonomy”, which provides a mapping of SWEBOK’s “generally accepted” knowledge areas to Bloom’s Taxonomy[25] to describe the expected knowledge of a software engineering graduate with four years of experience as “general-

ist” software engineer. “A software engineer with four years of experience is still at the beginning of their career [...] no topic is given a taxonomy level higher than Analysis¹⁰.”[4] This classification should be seen as early proposal of intended learning objectives¹¹, as stated: “please bear in mind that the evaluations of this Appendix should definitely only be seen as a proposal to be further developed and validated.”[4]

The SWEBOK version 3.0[112] defines a set of non-technical skills in the *Professional Practice* knowledge area. This knowledge area is divided into subareas *Professionalism, Group Dynamics and Psychology* and *Communication Skills* and is “[...] concerned with the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner.”[112] This version of the SWEBOK does not provide a mapping to taxonomy levels.

The *Software Engineering Competency Model* Software Engineering Competency Model (SWECOM)[14], as of 2014, is a project of the IEEE Computer Society to provide a description of competencies for software engineers. This competency model holds “A competent person has the skills needed to perform, at a given level of competency, the work activities assigned to him or her. *Knowledge*, in this competency model, is different from *skill*: knowledge is what one knows, while skill is what one can do. ”

Besides technical skills, the model contains:

- *cognitive skills (reasoning, analytical skills, problem-solving, innovation; may overlap and be combined; levels are not included)*
- *behavioral attributes and skills (aptitude, initiative, enthusiasm, work ethic, willingness, trustworthiness, cultural sensitivity, communication skills, team participation skills, and technical leadership skills; allow software engineers to contribute to desired outcomes effectively; others might be added)*
- *related disciplines (including but not limited to computer engineering, computer science, general management, mathematics, project management, quality management, and systems engineering)*
- *requisite knowledge (the intellectual basis for the SE profession, defined by the consolidated reference list in SWEBOK [112]).[14]*

The sixty defined *technical skills* are organized into five *Life Cycle Skill Areas* and eight *Crosscutting Skill Areas*. Each of those skills gets de-

¹⁰ Bloom’s Taxonomy[25] contains six levels in the cognitive domain: *Knowledge, Comprehension, Application, Analysis, Synthesis, and Evaluation*. Section 2.3.1 describes the revised version of this taxonomy that changed these levels and introduced a second dimension to describe learning objectives.

¹¹ Neither the term skill nor the term competency are used in the appendix.

composed into activities that a skilled person has to accomplish. *Software Process and Life Cycle Skills* form one of the *Crosscutting Skill Areas*.

The SWECOM defines five levels of competency: *Technician*, *Entry Level Practitioner*, *Practitioner*, *Technical Leader*, and *Senior Software Engineer*. “In general, a Technician follows instructions, an Entry Level Practitioner assists in performance of an activity or performs an activity with supervision; a Practitioner performs activities with little or no supervision; a Technical Leader leads individuals and teams in the performance of activities; and a Senior Software Engineer modifies existing methods and tools and creates new ones. [...]”[14]

The competency model does not prescribe knowledge levels or years of experience to reach one of the levels, but it delivers examples, e.g. “An individual who is competent at the Practitioner level [...] might have a master’s degree in software engineering or a related discipline, and would probably have more than five years of experience in the relevant skill areas.”

From an SE education perspective and having undergraduates’ intended learning objectives in mind, only the first two levels are of interest, as the other levels require years of experience in industrial practice.

The model defines at each activity which role, one or more of *Follows*, *Assists*, *Participates*, *Leads* and *Creates*, a given competency level might fill.

SWECOM provides worksheets for individual gap analysis to analyze available and needed skills and to plan improvement. Use cases are provided for different usages at an individual or organizational level.

With its competency levels the SWECOM draws on the *Software Assurance Competency Model*[283] of the Software Engineering Institute of the Carnegie Mellon University, which might serve as an example of a specialized competency model, in this case addressing the field of *Software Assurance (SwA)* defining an own specialized *Core Body of Knowledge (CorBoK)*. This competency model is focused on “employers of SwA personnel with a means to assess the SwA capabilities of current and potential employees.”[283] It aims at guiding to academic or training organizations, provide curricula guidance and direction for development and career planning of SwA professionals.

Based on and in reaction to the SWEBOK, Sedelmaier and Landes[243] proposed *Software Engineering Body of Skills (SWEBOS)*, an attempt to provide a “competency profile for software engineering,” which is focused on the non-technical soft-skills.

They acknowledge the focus on technical expertise in SWEBOK[112] and Software Engineering Education Knowledge (SEEK)[133] but criticize “Both handbooks give only superficial recommendations, if any,

of which soft skills a software engineer should have and what a particular soft skill exactly means. In addition, as pointed out above, there is no indication of which methods were used to derive the recommendations in the SWEBOK and the IEEE/ACM curriculum.” By using a data-driven scientific approach, they attempt to “foster a deep understanding of which competencies a software engineer must have. [...] to understand the semantics of context-sensitive soft skills in a software engineering context.” Context-sensitive soft-skills are explicitly delineated from generic soft-skills that are largely independent of SE and relevant for other disciplines too, e.g. presentation skills. Their work resulted in a set of competencies required for SE, each subset with a general definition and a list of indicators enabling an assessment if and to what extent a particular competence (set) is present. The identified competencies are:

- competencies for professional collaboration,
- communicative competencies,
- competencies for structuring one’s own way of working,
- personal competencies,
- competencies of problem awareness,
- competence to solve problems, and
- additional competencies

This approach does not make use of any defined levels.

According to Sedelmaier and Landes[243] “the three top soft skills in software engineering are:

- Comprehension of the complexity of software engineering processes and understanding of cause-effect relationships;
- Problem-awareness and the capability to develop creative solutions;
- Team competence including communication skills.”

Competencies demanded from academic programs are described in curriculum guidelines. To get an impression, what is demanded of SE curricula, the following section presents three current curriculum guidelines in the field of computer science and SE.

2.3.3 Existing Software Engineering (SE) Curriculum Guidelines

Today we find a number of curriculum guidelines both for undergraduate[95, 133] and graduate degree programs[95, 269].

	Knowledge Area	recommended contact hours	%
CMP	Computing essentials	152	32.5
FND	Mathematical and engineering fundamentals	80	17.1
PRF	Professional practice	29	6.2
MAA	Software modeling and analysis	28	6.0
REQ	Requirements analysis and specification	30	6.4
DES	Software design	48	10.3
VAV	Software verification and validation	37	7.9
PRO	Software process	33	7.1
QUA	Software quality	10	2.1
SEC	Security	20	4.3
	Σ	467	100.0

Table 2.7: SEEK Knowledge Areas[133]

This section examines learning objectives and competencies that curricula, satisfying those guidelines, should provide to students. Getting a better understanding of those objectives is prerequisite to locate the approaches introduced in this thesis/work inside such curricula.

The *Software Engineering 2014 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (SE2014)* is delivered by the *Joint Task Force on Computing Curricula*, pooling forces of the IEEE¹² Computer Society and the ACM¹³. It contains the SEEK defining “what every SE graduate must know.” Additionally, it provides guidelines to organize a curriculum with “ways this knowledge and the skills fundamental to software engineering can be taught in various contexts”[133]. An explicit goal of these guidelines is to “[...] be useful to computing educators throughout the world.”

The SEEK is divided into ten knowledge areas. These get summarized in table 2.7.

Each of the knowledge areas is subdivided into units, which again contain topics. The guidelines provide recommended contact hours for each knowledge unit. A Bloom taxonomy level¹⁴ is assigned to each of the topics to indicate the intended capability. For each of the

¹² Institute of Electrical and Electronics Engineers (IEEE)

¹³ Association for Computing Machinery (ACM)

¹⁴ The guidelines assign only the first three cognitive skill levels *Knowledge* (k), *Comprehension* (c) and *Application* (a) of the Bloom taxonomy.

subjects is mentioned, if that one is considered as an essential¹⁵ or desirable¹⁶ subject.

The knowledge area *PRO-Software process* is of particular interest for this thesis/work. The guidelines define this knowledge area as: “Software process is concerned with providing appropriate and effective structures for the software engineering practices used to develop and maintain software components and systems at the individual, team, and organizational levels. This knowledge area covers various process models and supports individual and team experiences with one or more software development processes, including planning, execution, tracking, and configuration management.”[133]

Compared to the *SE2004* version of the guidelines[132] the number of recommended minimum contact hours has remarkably increased¹⁷. The knowledge units *Project planning and tracking*, *Software configuration management*, and *Evolution processes and activities* were added. All but 2 of the 31 topics of the knowledge area are marked as *essential*. Out of the essential topics (n=29), one third is labeled with the taxonomy level *Knowledge* (n=10), one-third with the taxonomy level *Comprehension* (n=9), and the last third with taxonomy level *Application* (n=10).

These characteristics emphasize the increased importance attached to that knowledge area, that should provide a profound knowledge, going beyond just remembering (2/3), and that is to a remarkable extent ready for operation (1/3).

In addition to the SEEK knowledge areas the guidelines[133] define a set of “*qualities*.” Graduates of an undergraduate SE program should be able to:

- “Show mastery of software engineering knowledge and skills and of the professional standards necessary to begin practice as a software engineer.” (*professional knowledge*)
- “Demonstrate an understanding of and apply appropriate theories, models, and techniques that provide a basis for problem identification and analysis, software design, development, implementation, verification, and documentation.” (*technical knowledge*)
- “Work both individually and as part of a team to develop and deliver quality software artifacts.” (*teamwork*)
- “Demonstrate an understanding and appreciation of the importance of negotiation, effective work habits, leadership, and good

¹⁵ “The topic is part of the core.”[133]

¹⁶ “The topic is not part of the core, but it should be included in the core of a particular program if possible; otherwise, it should be considered part of elective materials.”[133]

¹⁷ from 13 in 2004 to 33 in 2014

communication with stakeholders in a typical software development environment.” (*end-user awareness*)

- “Design appropriate solutions in one or more application domains using software engineering approaches that integrate ethical, social, legal, and economic concerns.” (*design solutions in context*)
- “Reconcile conflicting project objectives, finding acceptable compromises within the limitations of cost, time, knowledge, existing systems, and organizations.” (*perform trade-offs*)
- “Learn new models, techniques, and technologies as they emerge and appreciate the necessity of such continuing professional development.” (*continuing professional development*)

Furthermore, the document provides a comprehensive set of curriculum guidelines. Curriculum guidelines 8 and 9 define “certain personal skills” that any curriculum should foster “primarily through practice.” Summarized shortly, students should be enabled to *exercise critical judgment, evaluate and challenge received wisdom* (think critically), *recognize own limitations* (and appreciate teamwork), *communicate effectively*, and *behave ethically and professionally*.

The guidelines demand a capstone project and describe intended characteristics of it.

Although the curriculum guidelines do not explicitly use the term *competency* to define learning outcomes, they describe intended acquired knowledge, skills and attitudes that should be used to accomplish goals in given contexts, hence, following the given definition at the beginning, they define *competencies* too.

In July 2016 the *German Informatics Society (GI)* published their updated recommendations for the design of undergraduate and graduate curricula of Computer Science (CS) programs[95].

As this is a recommendation for CS curricula and not specifically for SE curricula, the consideration of SE topics is remarkable minor. Compared to the *SE2014* guidelines, it provides much less guidance in this area. SE is described as one competency area of the *analysis, design, realization, and project management competencies* set.

Cognitive competencies are specified in the competency model using a modified version[43] of the *Taxonomy Table* introduced with the revised version of Bloom’s Taxonomy[12]. This competency model distinguishes between *high contextualization and complexity* and *low contextualization and complexity* at characterizing competencies and omits the lowest cognitive level *remember* of the revised taxonomy. The defined software process related competencies are classified at cognitive level *understand* (with *low contextualization and complexity*), at cognitive level *apply* (with *high contextualization and complexity*), and at cognitive level *analyze* (with *low contextualization and complexity*).

In addition to the cognitive competencies, the GI guidelines define a set of non-cognitive competencies whose acquisition should be fostered throughout the curriculum and that are expected to get acquired primarily implicitly in association with the cognitive competencies.

The GI guidelines do not provide a recommended number of contact hours per competence or knowledge area but provide a number of example curricula for orientation and guidance.

The GI guidelines stress that a course or capstone project is an essential characteristic of any curriculum to provide students with the opportunity to manage a complex endeavor and experience all phases of a software project.

The Graduate Software Engineering 2009 (GSWE2009): Curriculum Guidelines for Graduate Degree Programs in Software Engineering[269] “[...] primarily addresses the education of students for a professional master’s degree in SwE—that is, a degree intended for someone who is primarily interested in pursuing a career in the practice of SwE [...]. Typically, such students are already (a) professional software engineers employed by industry or government and who lack a formal graduate education in SwE, or (b) professionals in another field who are making a career change into SwE. In some cases, those students will be fresh graduates with a bachelor’s degree with little or no experience.”

In order to distinguish it from the IEEE/ACM guidelines¹⁸ for undergraduate curricula the authors state: “GSWE2009 expects much greater sophistication in student reasoning about SwE¹⁹ principles, and expects students to demonstrate their accumulated skills and knowledge in a more significant capstone experience (project, practicum, or thesis) than does SE2004. The courses, evaluations, and the capstone will generally be more demanding because GSWE2009 is a graduate curriculum [...] and GSWE2009 assumes that students enter the program with at least two years of relevant software development experience.”[269]

The GSWE2009 defines an own Core Body of Knowledge (CBOK) that is based on SWEBOK, SE2004, and *Systems Engineering* guidelines, adding topics that were not included in the SWEBOK at that time. CBOK, defined as “description of the fundamental or core skills, knowledge, and experience to be taught in the curriculum to achieve the outcomes,”[269] is subdivided into eleven knowledge areas that are again subdivided into knowledge units, which got Bloom’s Taxonomy[25] levels assigned.

¹⁸ SE2004 at that time

¹⁹ The abbreviation SwE stands for *Software Engineering*. It is used here to distinguish this discipline from *Systems Engineering* that is abbreviated SE in the context of that document.

“The CBOK knowledge units and their Bloom level designations were developed in such a way that the core could be covered in the equivalent of approximately 15 credit hours or approximately 200 contact hours (using a North American academic model). The core is designed to comprise a little less than 50% of the total credit hours recommended for a master’s degree. Hence, additional time and courses can be allocated to provide additional depth in the core areas (at higher Bloom levels) and to focus on a chosen application domain.”[269]

The assigned cognitive levels specify the minimum knowledge levels intended for all students in those knowledge areas. “Rather than create yet another SwE competency model, the team used the SWE-BOK as a widely-available, collaboratively-developed and thoroughly-vetted taxonomy.”[269]

Table 2.9 on the following page outlines the knowledge areas and their knowledge units. The high count of knowledge units labeled with the Bloom’s Taxonomy level *Application (AP)* or *Analysis (AN)* emphasizes two objectives. Firstly, the degree of sophistication, these guidelines aim at, is significantly higher than the intended learning outcomes of SE2014. Also, the guidelines are orientating towards practical skills and experience, which aligns well with the definition aforementioned. The knowledge areas *I—Software Engineering Management* and *J—Software Engineering Process*, which contain units of knowledge somewhat overlapping from the perspective of alternative classifications, are, in sum, provided with a portion of between 10% and 13% of the recommended contact hours. This high value shows that remarkable importance is attributed to these knowledge areas.

In addition to mastering the CBOK the GSwE2009 defines further learning outcomes, forming skills and attitudes expected from graduates after such a program. These additional outcomes are summarized in table 2.10 on page 53.

2.3.4 Conclusions

This section introduced the Original and The Revised Version of Bloom’s Taxonomy as well as definitions of skills and competency.

Three curriculum guidelines were presented. They provide valuable insights, which competencies are expected of graduates of curricula satisfying those guidelines, both in cognitive and non-cognitive competencies. These guidelines are designed oriented on outcomes and show that remarkable importance is attached to knowledge areas around *SE processes/methods*. This perception is confirmed by studies exploring industry’s expectations.[1, 242]

Among others, the skills to comprehend the complexity of software engineering processes, to understand cause-effect relationships,

	CBOK Knowledge Area / Knowledge Unit (KU)	Bloom Level	contact hours in % ²⁰
A	Ethics and Professional Conduct		1 – 2
	3 KUs, Bloom Levels: C(n=2), C/AP (n=1)		
B	System Engineering		2 – 3
	7 KUs, Bloom Levels: C(n=5), C/AP (n=2)		
C	Requirements Engineering		6 – 8
	8 KUs, Bloom Levels: C(n=1), C/AP (n=2), AP (n=4), AN (n=1)		
D	Software Design		9 – 11
	6 KUs, Bloom Levels: C/AP (n=1), AP (n=3), AP/AN (n=2)		
E	Software Construction		1 – 3
	3 KUs, Bloom Levels: AP (n=3)		
F	Testing		4 – 6
	5 KUs, Bloom Levels: C/AP (n=1), AP (n=3), AP/AN (n=1)		
G	Software Maintenance		3 – 4
	4 KUs, Bloom Levels: AP (n=4)		
H	Configuration Management (CM)		2 – 3
	5 KUs, Bloom Levels: C/AP (n=1), AP (n=3) ²²		
I	Software Engineering Management		7 – 9
	1 Software Project Planning	AP	
	2 Risk Management	AP	
	3 Software Project Organization and Enactment	AP	
	4 Review and Evaluation	C	
	5 Closure	C	
	6 Software Engineering Measurement	AP	
	7 Engineering Economics	C	
J	Software Engineering Process		3 – 4
	1 Process Implementation and Change	C/AP	
	2 Process Definition	C	
	3 Process Assessment	AP	
	4 Product and Process Measurement	AP	
K	Software Quality		3 – 4
	3 KUs, Bloom Levels: AP (n=3)		

Table 2.9: GSWE2009: Knowledge Areas And Knowledge Units[269]

LO	Description
1	"Master software engineering in at least one application domain, such as finance, medical, transportation, or telecommunications, and one application type, such as real-time, embedded, safety-critical, or highly distributed systems. That mastery includes understanding how differences in domain and type manifest themselves in both the software itself and in its engineering, and includes understanding how to learn a new application domain or type."
2	"Master at least one KA or sub-area from the CBOK to at least the Bloom Synthesis level."
3	"Be able to make ethical professional decisions and practice ethical professional behavior."
4	"Understand the relationship between SwE and SE and be able to apply SE principles and practices in the engineering of software."
5	"Be an effective member of a team, including teams that are international and geographically distributed, effectively communicate both orally and in writing, and lead in one area of project development, such as project management, requirements analysis, architecture, construction, or quality assurance."
6	"Be able to reconcile conflicting project objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems, and organizations."
7	"Understand and appreciate feasibility analysis, negotiation, and good communications with stakeholders in a typical software development environment, and be able to perform those tasks well; have effective work habits and be a leader."
8	"Be able to learn new models, techniques, and technologies as they emerge, and appreciate the necessity of such continuing professional development."
9	"Be able to analyze a current significant software technology, articulate its strengths and weaknesses, compare it to alternative technologies, and specify and promote improvements or extensions to that technology."

Table 2.10: GSwE2009: Additional Expected Outcomes[269]

to learn new models and technologies as they emerge and to develop a holistic “*software engineering mindset*” in general are repeatedly and widely stressed. “Underlying and enduring principles of software engineering should be emphasized, rather than the details of the latest or specific tools.”[133] As graduates enter the industry they are expected to be able to orient themselves in highly dynamic contexts in short time. This calls for highly transferable competencies.

2.4 DIGITAL GAME-BASED LEARNING (DGBL)

Digital games in general and serious games in special have become a huge market in industry and a remarkable field of academic research[228]. By looking at sales statistics, it becomes apparent that digital games have characteristics that are highly attractive to people of all age groups.

In 2015 consumers in the U.S. spent \$23.5 billion on gaming, including content (\$16.5 billion), hardware (\$4.9billion), and accessories (\$2.1 billion).[82]

In 2012 33% of surveyed company representatives in a German survey stated, they are using serious games for training purposes. 51% reported being using simulations for that purpose.[23]

Dondi and Moretti note “Game-based learning has become an issue of great interest. Many conferences, papers and books are focused on presenting the advantages of using or better adopting game-based learning for supporting motivation in learning and for improving skills and competences.”[74]

Before we examine, what—according to researchers’ and game designers’ opinion—makes digital games and their combination with learning that exciting, we have to clarify some terminology.

2.4.1 Terminology

2.4.1.1 Play And Game

According to Huizinga[111] “play is older than culture [...] We can safely assert [...] that human civilization has added no essential feature to the general idea of play. [...] even in its simplest forms on the animal level, play is more than a mere physiological phenomenon or a psychological reflex. It goes beyond the confines of purely physical or purely biological activity. It is a *significant* function—that is to say, there is some sense to it. In play, there is something ‘at play’ which transcends the immediate needs of life and imparts meaning to the action. All play means something.”

Van Eck[295] agrees “play is a primary socialization and learning mechanism common to all human cultures and many animal species. Lions do not learn to hunt through direct instruction but through modeling and play. Games, clearly, make use of the principle of play as an instructional strategy.” Play serves as powerful tool for collaboration and mediator for learning throughout a person’s life.[87, 227]

Huizinga[111] describes six characteristics of play:

1. *Voluntary*, “in fact freedom”: “play to order is no longer play: it could at best be but a forcible imitation of it.”

“Anyone who makes a distinction between games and education clearly does not know the first thing about either one.”

—Marshall McLuhan (quoted by Marc Prensky)

“The whole truth regarding play cannot be known until the whole truth regarding life itself is known.”

—Lehman and Witty (quoted by Jesse Schell)

2. *Pretend*: “play is not ‘ordinary’ or ‘real’ life. It is rather a stepping out of ‘real’ life into a temporary sphere of activity with a disposition all of its own. [...] connected with no material interest [...].”
3. *Immersive*: “absorbing the player intensely and utterly.”
4. *Limited in time and place*: “It is ‘played out’ within certain limits of time and place. It contains its own course and meaning.”
5. *Order*: “it creates order, *is* order. Into an imperfect world and into the confusion of life it brings a temporary, a limited perfection.”
6. *Social*: “promotes the formation of social groupings which tend to surround themselves with secrecy and to stress their difference from the common world by disguise or other means.”

Botturi and Loh[37] explain “playing has a special feature [...] a playful dimension [...] playing is not something that we do distinctly apart from daily life. It is a modality of doing things, a *mode* of human experience, a sort of envelope of what we do that give a specific different hue to the activities that we perform. This mode of experience is natural to children, while it is more difficult to adults.”

As Michael and Chen[166] note “Serious games often violate one of the six characteristics listed above in that they aren’t always voluntary activities. Trainees may indeed be ordered to play a particular game as part of their training. This doesn’t mean that the serious game cannot be fun.”

To define the term *game* is a difficult exercise. Botturi and Loh state “Defining the concept of game can be [...] elusive. We all probably have a quite clear idea of what a game is based on our experience of playing games, maybe as kids, and have a number of good examples. But how would we define it?”[37] The attempts to find a definition of *game* fills whole book chapters, e.g. [37, 138, 237]. As there is not the room to describe all those attempts and kinds of definitions in this thesis, a pragmatic one was chosen.

Schell[237] examined a set of definitions and distilled ten quality attributes of games:

- Q1: Games are entered *willfully*.
- Q2: Games have *goals*.
- Q3: Games have *conflict*.
- Q4: Games have *rules*.
- Q5: Games *can be won and lost*.

“A game is a
problem-solving
activity, approached
with a playful
attitude.”
—Jesse Schell (2008)

- Q6: Games are *interactive*.
- Q7: Games have *challenge*.
- Q8: Games can create their own *internal value*.
- Q9: Games *engage* players.
- Q10: Games are *closed, formal systems*.

He defines: “A game is a problem-solving activity, approached with a playful attitude.”[237] In his explanation Schell proves that this definition takes all of those quality criteria into account.

We can see that there is quite a significant intersection between the characteristics of *play* and *game*, but that they are not the same. Prensky terms *games* just as “organized play”[218].

2.4.1.2 DGBL, Serious Games, And (Un-)Related

In 1975 Abt introduced the use of *serious games* and states “We are concerned with serious games in the sense that these games have an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement.”[5]

Without limiting that approach to games that were actually designed to provide the educational effect, he proposed to use them for this purpose. The educational purpose of a game, not designed for this effect, may be added by educators utilizing the game to achieve learning objectives.

The term *serious games* applied to digital games is attributed to Sawyer[40, 73], who published a white paper on the potential benefit of using digital games for policy making in 2003.

Since Abt’s definition, which is still considered to be mostly valid, several others were proposed. “The definition of the term ‘serious game’ often varies depending on who uses it and in what context. [...] but the great majority share the core statement that serious games are games which are used for more than just mere entertainment.”[40] This way, serious games also include a broad range of games, e.g. games used to distract patients undergoing painful therapies.

The term *Digital Game-Based Learning (DGBL)* was coined by Prensky with his identically named book in 2007. He defines DGBL as “any marriage of educational content and computer games.”[218] Again this is a very broadly defined definition, including a lot of use cases, games used and designed specifically for the educational purpose as well as *commercial off-the-shelf (COTS)* games used for educational purposes. This DGBL definition “is identical to the modern use of ‘serious games’ for computer and video games with/for educational purposes.”[40]

The terms and fields of *serious games* and *DGBL* are related to *e-learning* and *edutainment*, and *entertainment education*.

E-Learning encompasses all forms of learning, at which digital media support presentation and distribution of learning content and/or human communication[143].

E-Learning is not bound to a particular medium and does not necessarily aim at entertainment and fun. It provides remote, distributed, and asynchronous learning. As such, *serious games* and *DGBL* can be seen as a subcategory of e-learning. "E-Learning is a concept that has been and still is as popular as serious games and is researched in various disciplines like psychology, pedagogy or computer and information science." [40]

Edutainment, short for *education through entertainment*, is a term that got popular in the 1990s with "multi-media" appearing on personal computers. With regards to edutainment Michael and Chen state "[...] edutainment [...] most often refers to video games with overtly educational aims, specifically for preschoolers and new readers. Serious games, however, [...] move past the limited focus of edutainment to encompass all types of education and at all ages. Edutainment titles are considered a subset of the overall topic of serious games." [166]

Entertainment education is a more general term used to encompass the combination of entertainment and education, that is not bound to specific media. Models of this type "[...] suggest a sweet spot to perfectly blend entertainment and education together in one [...] experience." [252] As such, serious games form a subset of *entertainment education*.

Another omnipresent term popular nowadays is *Gamification*. Deterding et al. [70] summarize " *Gamification* refers to

- *the use* (rather than the extension) *of*
- *design* (rather than game-based technology or other game-related practices)
- *elements* (rather than full-fledged games)
- *characteristic for games* (rather than play or playfulness)
- *in non-game contexts* (regardless of specific usage intentions, contexts, or media of implementation)."

Taking this definition into account *gamification* does not form a superset or subset of *DGBL* or *serious games*. Instead of utilizing games, *gamification* makes use of *design elements characteristic for games*. As such, this term is rather loosely related to *DGBL*, as both share game design elements, but concepts of *gamification* may be combined with *DGBL* and *serious games* to create learning experiences. Typical elements used to gamify non-game contexts are "reward structures, positive reinforcement, and subtle feedback loops alongside mechanics

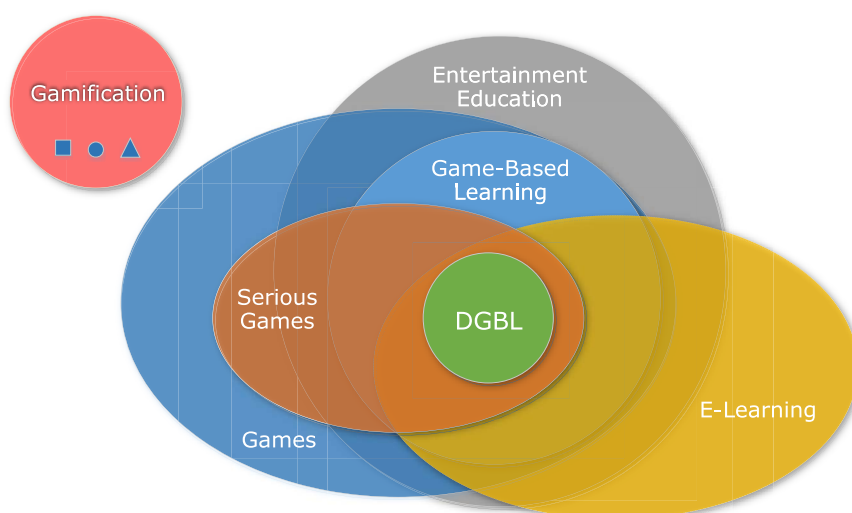


Figure 2.4: DGBL, Serious Games and Related Concepts [adapted from 40, 41]

like points, badges, levels, challenges, and leaderboards.”[314]

Figure 2.4 illustrates the relationships between these described concepts.

2.4.2 Games And Learning

This section is concerned with the following four questions:

1. What characteristics contribute games²³ to learning—why are they beneficial?
2. How much fun, or entertainment, is needed for game-based learning—and how should it be integrated?
3. What are potential drawbacks of game utilization in education?
4. How does DGBL fit with today’s learners?

“The most important learning skills that I see children getting from games are those that support the empowering sense of taking charge of their own learning.”
—Seymour Papert (1988)

²³ When talking about games, it has to be recognized, that there obviously is not the one game—as there is not the one book unifying all good characteristics attributed to books. Games are (just) digital media. Each game has its own—potentially unique—characteristics. That is why claims resulting from studying one game, or even a set of games, should be generalized only with caution. When talking about games in this section, they should be perceived as good games or games done right. Characteristics described in this section describe the potential of (well done) games—characteristics that (of course) not every game might be delivering.

2.4.2.1 *What Characteristics Contribute Games to Learning—why Are They Beneficial?*

Games and learning share some essential characteristics. “Like games, learning is an interactive process, challenges the learners and has more or less explicit rules on how to acquire new knowledge or skills.”[40]

Van Eck[295] states “Games are not effective because of what they are, but because of what they embody and what learners are doing as they play a game.” They “embody well-established principles and models of learning. For instance, games are effective partly because the learning takes place within a meaningful (to the game) context. What you must learn is directly related to the environment in which you learn and demonstrate it; thus, the learning is not only relevant but applied and practiced within that context. Learning that occurs in meaningful and relevant contexts is more effective than learning that occurs outside of those contexts, as is the case with most formal instruction. Researchers refer to this principle as *situated cognition* and have demonstrated its effectiveness in many studies over the last fifteen years.”

Besides *situated* and *problem-based learning*, other educational approaches are utilized in DGBL. Games may provide an *anchor*, in the sense of *anchored instruction*. Obviously interactive games provide *learning by doing* and *discovery learning*, *guided*, or *scaffolded*, by gameplay, rules of the game and/or *intelligent tutors*. Without consequences for real life, they invite to *learning from failures*. All these educational approaches and underlying learning theories got introduced in section 2.2 on page 19. Digital games provide a natural environment for those learning approaches.

Digital games employ a visual medium that is *interactive* and provides *immediate feedback*. These characteristics enable several beneficial features.

Players recognize their own actions to be effective in the game world. This provides a pleasurable sense of *control* and *self-efficacy*²⁴. It invites learners to accept taking control and responsibility for their own learning.

Interaction and immediate feedback enable a cycle of “probing the world (doing something); reflecting in and on these actions and, on this basis, forming a hypothesis; reprobing the world to test this hypothesis; and then accepting or rethinking the hypothesis.”[93]

“Games thrive as teaching tools when they create a continuous cycle of cognitive disequilibrium and resolution (via assimilation or ac-

²⁴ *Self-efficacy* is a term coined by psychologist Bandura[17]. It describes one’s confidence in one’s own ability to master tasks, to solve problems and to reach set goals.

commodation) while also allowing the player to be successful.”[295]

Games invite for *engagement* by providing *fun* and—even more—*challenge*. Gee[93] calls digital games “*pleasantly frustrating*.” Most digital games need remarkable effort to enjoy them, which makes playing them *hard fun*. Papert[196] notes “kids who talk about ‘hard fun’ and they don’t mean it’s fun in spite of being hard. They mean it’s fun because it’s hard. Listening to this and watching kids work at mastering games confirms what I know from my own experience: learning is essentially hard; it happens best when one is deeply engaged in hard and challenging activities. [...] The fact is that kids prefer things that are hard, as long as they are also interesting.” Gee[93] supposes “The key is finding ways to make hard things life enhancing so that people keep going and don’t fall back on learning only what is simple and easy.” In an ideal state, games may provide *flow*²⁵.

Games are great motivators by providing *rewards*, *gratification* and *competition* through points, badges and leaderboards on the one hand and *social interaction* through *collaboration* and the *communication* of game experiences on the other hand.

2.4.2.2 *How Much Fun, or Entertainment, Is Needed for Game-based Learning—and How Should It Be Integrated?*

How much fun or entertainment is needed to support successful learning? According to Ritterfeld and Weber[229], there are basically three theoretical assumptions. These are illustrated in figure 2.5 on the following page. The first one is *linear positive*: the more fun, the more learning. Fun acts as a facilitator (a). The second assumption, *linear negative*, represents the complete opposite. The more fun, the less learning. Under this assumption, fun is a distractor interfering learning (b). The third variant, *inverse U-shape*, combines first and second variants. Learning is facilitated by fun up to a certain point but interfered if even more entertainment is added. According to this model, an educator has to find the *sweet spot*[229], where fun maximizes learning, to build an ideal learning experience. The majority of game designers and DGBL researchers favor the first or third assumption.[41]

Besides the amount of entertainment, its timing is of interest at designing serious games. When should entertainment enrich a game to facilitate learning? Ritterfeld and Weber identified three paradigms of entertainment-education. These are illustrated in figure 2.6 on the next page.

²⁵ *Flow*[64] describes a mental state, where challenges presented and the ability to solve them are almost perfectly matching. People in flow state feel a complete energized focus in an activity combined with a high level of enjoyment and focus. They lose track of time and worries during this experience.[53, 64, 218]

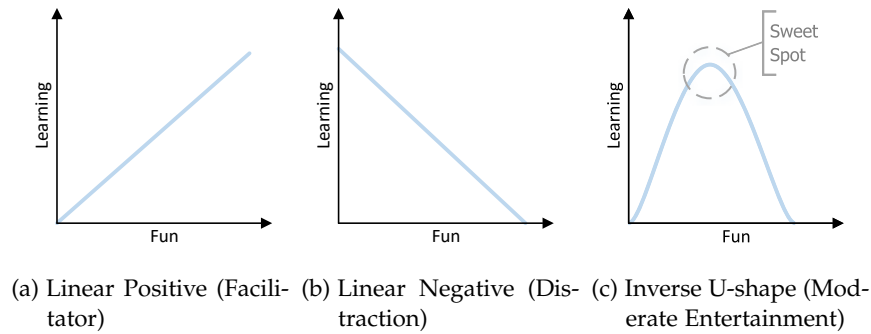


Figure 2.5: Relationship Between Fun And Learning: Different Theoretical Assumptions [adapted from 229]

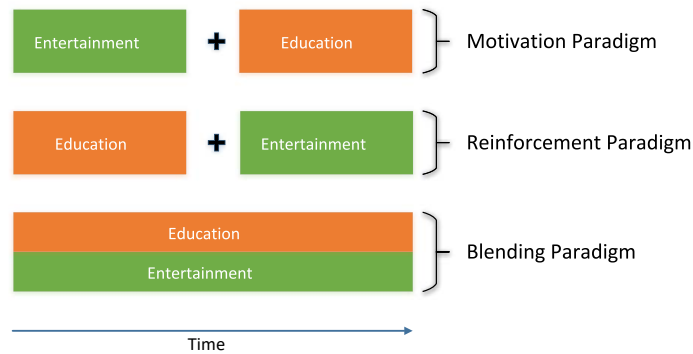


Figure 2.6: Paradigms of Entertainment-Education [adapted from 229]

In the *motivation paradigm*, elements of entertainment are used to gain attention and prepare for learning. This implies “that the content alone would not be a strong enough attractor to ensure processing, and it requires enrichment through entertainment.”[229]

The *reinforcement paradigm* utilizes entertaining elements as rewards for successful learning. “A reinforcement strategy is applied in most educational games through scores, virtual money, fun animations, or the reward of progress in the video game play.”[229]

Both extrinsic approaches may unintentionally result in a motivational decrease. “If individuals are intrinsically motivated, they do not have to be preached to. Nor do they need incentives in order to perform a task. In these individuals, reinforcement may even reduce intrinsic motivation in the long run.”[229]

In the last *blended paradigm*, entertainment and education are not sequenced but integrated. The learning approach itself is designed to be intrinsically “entertaining, i.e. the enjoyment of mastery in the game is equivalent to the enjoyment of the acquisition and use of knowledge and skills.”[40]

2.4.2.3 What Are Potential Drawbacks of Game Utilization in Education?

At the same time that games provide such rich learning environments, they may carry disadvantages²⁶ too.

Especially for learners primarily seeing just a fun activity in a learning game, such activity may lead to a media-induced decreased *amount of invested mental effort (AIME)*[234] interfering learning outcomes. Learners that are accustomed to *twitch speed* games always in need for speed may run in danger too, as Prensky cites Stoll, “substitute quick answers and fast action for reflection and critical thinking.”[218]

The right balance of fun, or entertainment, and challenge may keep the necessary focus and engagement of the learner. This emphasizes the “necessity for a blended learning experience which seamlessly integrates enjoyment and learning and presents the learning content as something which is neither external to the game nor a juxtaposition of entertaining sequences and educational material.”[40] As Papert emphasizes “talking about games and learning is an important activity.”[196]

Games and simulations, which are based on models (cf. section 2.1 on page 16), may lead to misconceptions in learning due to reductions and “simulation shortcuts.”[166] At this point, the careful embedding of the serious game into the whole learning scenario and environment including debriefing actions is of enormous importance. “If the educators are aware of these ‘flaws’, they can use these as links to address questions that are posed or left open by and in these games. They serve as an ideal anchor for complementing educational activities.”[40]

2.4.2.4 How Does DGBL Fit With Today’s Learners—born Digital, Digital Native, Net Generation?

Prensky states that frequently bemoaned short attention spans of a growing group of learners are due to a lack of engaging activities in old ways of learning. “They certainly don’t have short attention spans for their games, movies, music, or Internet surfing. [...] The kids will master systems [...] and read far above their grade level—when the goals are worth it to them.”[217]

This emerging group of learners is often referred to as *Digital Natives* or *Net Generation*, “because they are ‘native speakers’ of the digital language of computers, video games and the Internet—loosely, those born after the 1980s.”[37]

Van Eck[295] states that this group has become disengaged with traditional instruction because it lacks important characteristics: “They

²⁶ To keep this thesis focused, only the impact of disadvantages on learning environments in the narrower sense are considered here. An—often passionate—discussion of game effects on unwanted violent behavior is beyond the scope of this thesis. The same applies to addictive consumption of digital games.

require multiple streams of information, prefer inductive reasoning, want frequent and quick interactions with content, and have exceptional visual literacy skills—characteristics that are all matched well with DGBL.”

2.4.3 *Employment examples*

DGBL is applied in many different contexts and domains. To keep this thesis focused, it will not list all the scenarios, where DGBL and serious games were deployed with success. There are already both, an enormous body of literature on this topic as well as comprehensive databases²⁷ that the interested reader may consult.

Instead, this thesis focuses on the utilization of DGBL in SE education, i.e. in software process/method education. Section 2.5 on page 68 summarizes attempts in this field.

2.4.4 *Typology of Serious Games*

Several attempts to classify serious games exist today. A common attempt is to classify by educational content and application area of games.[166] Such an approach results in long lists of game types with a high number of subtypes.

But as Breuer and Bente[40] state “Genre definitions are always controversial” because such an attempt is “ignoring the fact that there are numerous genre hybrids [...]” and “genres are not usually categories creators of games think in when designing and producing games.”

Ratan and Ritterfeld[ratan2009classifying] proposed a classification in 4 different dimensions: (1.) Primary educational content, (2.) Primary learning principle, (3.) Target age group, and (4.) Platform. In their study, they classified 612 games.

Breuer and Bente propose the use of labels or tags to “overcome the problem of static and either incomplete or redundant genre systems.”[40] In section 4.7.11 on page 245 this approach is used to classify games developed in the context of this research work.

Another approach to provide flexibility is taken by Djaouti et al.[66]. They predefined sets of categorized *bricks*, representing some kind of category label for gameplay characteristics, that can be combined to larger *meta-bricks* and assigned to games²⁸.

Prensky[218] provides a categorization by positioning a game along different dimensions. This approach is used in section 4.7.11 on

²⁷ e.g. <http://www.socialimpactgames.com/>, <http://serious.gameclassification.com/>,
<http://www.gamesforchange.org/>, <https://gamesandimpact.org/games/>,
<http://studies.seriousgamesociety.org/>

²⁸ The result of their classification schema[66] can be found at <http://www.serious.gameclassification.com/>.

page 245 to categorize the *Simulation Game* developed in the context of this research work.

2.4.5 Challenging Evaluation

The potential for DGBL seems to be tremendous. This section examines what types of evaluations were used so far, what types of evaluation are recommended, and what makes the evaluation of DGBL approaches challenging.

De Freitas and Liarokapis[87] report “While arguments against serious games have centred upon a lack of empirical evidence in support of its efficacy, two large studies in the UK and US respectively have demonstrated positive results in large sample groups, in one study [...] considerable efficacy of game-based approaches over traditional learning techniques were demonstrated [...], while in another study [...] behavioural change in children with respect to medication adherence was proven in clinical trials. [...] the power of ‘immersive experiences’ is proving more engaging and motivating than standard approaches to training and education and more evidence of this efficacy is growing in the literature.”

Methods that have been employed so far to analyze games and their outcomes range from qualitative analysis, e.g. utilizing questionnaires, where learners self-report their experiences and perceived knowledge gain, to elaborated quantitative empirical research designs employing control groups and pre-/post-test designs. Besides knowledge or skill acquisition, games and their impacts have been analyzed along a number of dimensions, including motivational aspects and invested effort.

Connolly et al.[61] state “Despite the optimism about the potential of games for learning, several authors have noted that there has been a dearth of high quality empirical evidence to support these claims.” They call for “hard evidence to support these claims.” Points of criticisms in the field of DGBL/serious games research include that “literature on games is fragmented and lacking coherence” and that “there has been less in the way of hard evidence to support these claims.”[61] In their systematic literature review Connolly et al.[61] state with regards to knowledge acquisition: “There were few RCTs²⁹ and the evidence they provided about the impact of games was mixed.” While some of the reported game deployments resulted in improved performance other found no difference in performance of students using an online game and others using computerized flash cards, “although students preferred learning with the game and

“Games are not effective because of what they are, but because of what they embody and what learners are doing as they play a game.”
—Richard van Eck (2006)

²⁹ Randomized Controlled Trial (RCT)

enjoyed it more.”

Van Eck[295] criticizes evaluation approaches solely oriented towards quantitative empiricism: “These empirical studies are only part of the picture. Games are effective not because of what they are, but because of what they embody and what learners are doing as they play a game. Skepticism about games in learning has prompted many DGBL proponents to pursue empirical studies of how games can influence learning and skills. But because of the difficulty of measuring complex variables or constructs and the need to narrowly define variables and tightly control conditions, such research most often leads to studies that make correspondingly narrow claims about tightly controlled aspects of games (e.g., hand-eye coordination, visual processing, the learning of facts and simple concepts).”

Johnson states a little provoking: “When I read these ostensibly positive accounts of video games, they strike me as the equivalent of writing a story about the merits of the great novels and focusing on how reading them can improve your spelling.”[130]

As games represent digital media, each with its unique characteristics, results of studies are generalizable only with caution. Breuer[41] mentions that external validity, a prerequisite to experimental design, cannot be guaranteed since, different from other static media like books or movies, each run of a game may proceed differently—actually be unique. Dynamics and interactivity of digital games may provide different experiences resulting in different treatments of participants.

Connolly et al.[61] note “it is not always clear what a suitable control comparison would be [...]” This is especially true for attempts, where serious games are pioneering in new fields, which are missing generally approved learning material that could be used in control groups.

In general, the knowledge acquisition as an outcome of a serious game is measured after the gameplay by utilizing oral or written tests. This approach has several flaws. Learners may be less intrinsically motivated at playing a game by the fact that they are confronted with the same pressure to perform as in traditional teaching approaches.

Additional to this, such tests may query factual knowledge but to a much lesser extent practical and procedural competencies, creativity and productivity of learners.[41] This calls for an “effective and unobtrusive assessment methods for digital game-based learning need to be developed and evaluated to monitor not only the learning outcomes but also the learning process.”[40]

2.4.6 Conclusions

Sales statistics prove that digital games are highly attracting people of all age groups.

Finding a mutually agreed definition of *Game* proves to be a challenging exercise filling whole book chapters. Games exhibit a number of attributes qualifying them as *Game*. Shell's ten quality attributes of games got presented as one example of scoping and defining games. He[237] defines "A game is a problem-solving activity, approached with a playful attitude" and proves that his definition explains all of his defined quality attributes of games. Using *playful* as an attribute, this definition points towards a strong connection between *Play* and *Game*. Both share characteristics but are not the same (cf. section 2.4.1.1 on page 55). Prensky calls games "*organized play*." [218]

Games and learning share essential characteristics. Games (may) embody well-established principles and models of learning. As van Eck states "[...] DGBL can be implemented most effectively [...] by attending to these underlying principles." [295] DGBL fits well with the demands of a new learning generation and provides a vast number of characteristics supporting learning.

The combination of learning and entertainment/fun in digital games requires a careful consideration of both the amount and timing of introduced entertainment at designing a serious game. A balanced *sweet spot* has to be found where entertainment/fun ideally maximizes learning. Adding more entertainment on top may impede learning and result in students seeing just a fun activity in a learning game, which may cause a media-induced *decreased amount of invested mental effort (AIME)* [234] interfering learning outcomes. The *blended paradigm* (cf. figure 2.6 on page 62), where entertainment and education are not sequenced but integrated and "the enjoyment of mastery in the game is equivalent to the enjoyment of the acquisition and use of knowledge and skills," [40] seems to be the most promising approach for the intended *Simulation Game* introduced in section 4.7 on page 226.

The approaches to evaluate the success of DGBL are not without controversy. To build and evaluate a DGBL approach to support SE education demands a careful consideration of the underlying circumstances as well as the specific characteristics and challenges associated with the evaluation of digital learning games.

2.5 DGBL IN SE EDUCATION—EXISTING SOLUTIONS

A number of secondary studies summarize developments of simulation and DGBL in SE education[61, 126, 200, 202, 301] and provide a good starting point for investigations in the field. These studies were examining different aspects.

	[61]	[301]	[200, 202]	[126]	this thesis ^a
Review method	narrative, auto-mated	SLR	SLR	SLR	manual
Period	1996-2006	1990-2008	1999-2009	open-2013	2014-2016
Simulation games	Y	Y	Y	Y	Y
only computer-based	Y	N	Y	Y	N
non-game simulations	N	Y	N	Y	N
number of papers	21	16	16	42	17
number of simulators and games	12	12	8	15 ^b	13

Table 2.11: (Systematic) Literature Reviews of Simulation and DGBL in SE Education (adapted from Jiang et al.[126])

^a cf. appendix F on page 377

^b only simulators

Table 2.11 summarizes characteristics of these literature reviews. The review of most recent literature done for this thesis to catch most recent activities (cf. appendix F on page 377) was added. Results of the literature reviews naturally overlap to some extent. Jiang et al.[126] provide a rigorous and the most current Systematic Literature Review (SLR). Since their study examined computer-based simulation games as well as non-game simulations, their review covers the field of interest for this thesis. Results of their review combined with the review done for this thesis, covering the years after their SLR are presented in a timeline in Table 2.12 on page 70. The timeline was extended with some Ph.D. theses that were not covered by SLRs. The timeline proves an ongoing interest and research activity in the field. Recurring entries of simulation games indicate the effort affiliated with the development and evaluation of approaches—it may take several years from initial concept descriptions till the implementation and evaluation of approaches. Some implemented games or environments, i.e. *SimSE*, *SESAM*, and *AMEISE*, show an impressive body of work of several researchers covering long periods, both for refinement of initial versions as well as the evaluation of taken approaches.

Besides the utilization of DGBL, pure non-game-based rather industrial simulation models were used[210–212, 231] in training and education sessions. Since those attempts differ to some extent in their

approach and are lacking the benefits of utilizing DGBL, they are not examined here and are omitted in the timeline for the sake of brevity. Since the year 2000, such non-game simulations were mainly replaced by game-based approaches.[126]

Several attempts to utilize DGBL in SE education have already been made. Some of them were focused on specific, rather isolated, SE knowledge areas, others were addressing the area of SE processes and SE management. For the context of this thesis, the latter are of particular interest. Before those studies get presented, examples of other approaches get described to provide a more complete overview of the field.

Wang et al.[299] introduced a lecture quiz approach, where the lecturer hosts a set of quiz questions that get answered by students via their mobile devices in real time. Results are presented and provide triggers for interaction in a lecture. This approach is somewhat similar to the freely provided *kahoot*³⁰ blended learning system. In the context of this thesis, *kahoot* gets used to design interactive lectures in *phase 1* of the proposed *Integrated Approach* (cf. chapter 4 on page 179). Such a quiz approach alone does not provide the necessary explorative experience of running an SE endeavor striven for in this thesis.

A number of pure card or board games have been implemented and used. Taran's game[276] deals with risk management issues. *Problems and Programmers*, a card game by Baker et al.[15, 16], introduces students to the waterfall lifecycle. A card game by Soska et al.[261] enables students to learn concepts of software test. *GetKanban* is a board game introduced by Heikkilä et al.[108] to promote concepts of *Kanban*. Ganesh's board game *HardChoices*[89] introduces students to the SE concept of *Technical Debt*.

Card and board games, primarily designed to be played in a group of students, may provide a highly interactive learning environment promoting social aspects and discussion. The majority of card or board games is addressing single aspects of SE.

A number of studies are focused on requirements engineering. Hainey et al.[104, 105] provide *SDSim*, an example of utilizing DGBL to facilitate learning of requirements collection and analysis.

TREG of Vega et al. [296] utilizes the virtual 3D-world *SecondLife* to address the same topic.

qGame, introduced by Knauss et al.[145], is focused on requirements engineering too—but in a special way since it promotes a special concept of “*Software Quantum Metaphor*” to describe the flow of

³⁰ <https://getkahoot.com/how-it-works>

		literature reviews			
2016	Nassal's PMG (modeling human behavior)[173], <i>GSDgame</i> [293], SimSE (utilized for study of game design patterns)[86], <i>Smart Decisions</i> [51], <i>GetKanban</i> v4.0[108]				
2015	<i>Nassal's PMG</i> [172], <i>Zeid's Global Management</i> [307], <i>DesignMPS</i> [52], SPIAL (adoption issues)[205], <i>PMG-2D</i> [153], Essence Simulation Game (concept description)[214], Jiang et al. SLR[126]				
2014	Nassal's PMG (general framework)[171], <i>GSDSim</i> [185], <i>InspectorX</i> [216], SPIAL (evaluated using UGALCO framework)[204], <i>HardChoices</i> [89],				
2013	qGame (supporting concept of Quantum Metaphor)[239], SPIAL[203]				open-2013 Jiang et al. (2015)[126]
2012	<i>SPIAL</i> [201, 202], AMEISE (experience report)[35]				
2011	AMEISE (experience report)[34], SDSim (evaluation)[105], <i>Simsoft</i> (PhD thesis)[49, 50], Peixoto et al. SLR [200]				
2010	<i>SDSim</i> (PhD thesis)[104]				
2009	von Wangenheim and Shull SLR[301], <i>TREG</i> [296], <i>XMED</i> [298], SimSE (multi-site evaluation)[182]			1999-2009 Peixoto et al. (2011, 2012)[200, 202]	
2008	<i>qGame</i> [145], SimSE (transformation of process profile to SimSE model for validation)[110]		1990-2008 von Wangenheim and Shull (2009)[301]		
2007	<i>MO-SEProcess</i> [306], Connolly et al. SLR including concept of SDSim[62], SimSE (comprehensive evaluation)[181]				
2006	<i>SimVBSE</i> [125], Incredible Manager (extended evaluation)[18], SimSE (PhD thesis)[174]	1996-2006 Connolly et al. (2007)[62]			
2005	<i>SimjavaSP</i> [250], SimSE (first evaluation)[177], SimSE (model description)[179]				
2004	<i>SimSE</i> [176] SimSE (work in progress)[175] <i>Incredible Manager</i> [67]				
2003	<i>AMEISE</i> [167],				
2001	SESAM (elaborated deployment concept)[161]				
2000	OSS[249], SESAM (project and QA model description)[77]				
1999	SESAM (QA model description)[76]				
1998	SESAM (introduction of QA model)[160]				
1996	SESAM (project report)[157]				
1994	<i>SESAM</i> [69]				
1992	SESAM (prototype, WIP)[156]				
...					

years 1993, 1995, 1997, and 2002 without game-based entries were omitted for brevity,

years 2014-2016 contain non-simulation game-based studies and a SLR to provide a comprehensive overview of recent activities in the field (cf appendix F on page 377)

Table 2.12: Timeline of Simulation-Based DGBL Approaches in SE Education Focused On SE Process/Methods

requirements through the software development process. Recently this game was used to promote *requirements compliance* as a measure of project success[239].

DesigMPS, introduced by Chaves et al.[52], is focused on software process but from a different perspective. This serious game is concerned with the modeling of software processes (Software Process Modeling (SPM)).

In *XMED*, introduced by von Wangenheim et al., players can “exercise the application of software measurement in the context of project management.”[298]

With *ERPsim*, Utesch et al.[291, 292] report the utilization of an online simulation game based on the SAP Enterprise Resource Planning (ERP) system to train skills like “decision making, analysis, strategy development, data processing and presentation”[292] of future students.

Cervantes et al. introduced *Smart Decisions*, a serious game addressing the area of systematic architectural software design.

Recently a number of serious games are focusing on the area of Global Software Development (GSD), addressing challenges like communication overhead caused by different time zones, cultural diversity of team members, location barriers, as well as issues. These games include Zeid’s global management game[307], *GSDgame* introduced by Valencia et al.[293], and *GSDSim* introduced by Noll et al.[185].

2.5.1 Existing Simulations and Games for SE Process/ Method Education

Several attempts to provide software process, SE method, knowledge to students already exists. This section describes existing approaches.

Open Software Solutions (OSS) introduced by Sharp and Hall[249] in 2000 is a “multimedia simulation of a software house Open Software Solutions (OSS). The student ‘joins’ OSS as an employee and performs various tasks as a member of the company’s project teams.” Developed to present a case study, with approximately 80 hours of study, this approach addresses postgraduate distance education. OSS includes audio, animations, and rich graphics. The learner takes a rather passive “observer” role, may look into provided documents, listen to meetings and discover task descriptions. This approach represents rather a kind of introductory tutorial into the areas of SE. The players cannot interactively make their own decisions to control the

	SimSE[174]	SimjavaSP[250]	SESAM[76, 77, 161], AMEISE[167]	Incredible Manager[18]	OSS[249]	MO-SEProcess[306]	SimVBSE[125]	Simsoft[49]	Nassal's PMG[172]	SPIAL[201, 202]	PMG-2D[153]
single player (S) or multiplayer (M) game	S	S	S	S	S	M	S	M	S	S	S
collaboration and competition of players	-	-	-	-	-	-	-	^a	-	-	-
team, course and lecturer dashboards	-	-	-	-	^b	-	-	-	-	-	-
player in the role of a project manager	+	+	+	+	o	o	+	+	+	+	+
player(s) with different roles or tasks	-	-	-	-	+	+	-	-	-	o	-
enabled for different software processes	+	-	+	-	-	-	-	-	+	+	-
number of available predefined SE process/method models	6	1	3 ^c	1	1	1	1	1 ^d	1	1	1 ^e
domain specific language for modeling	-	-	^f	-	-	-	-	-	^g	o	-
rule editor for modeling	+	-	-	-	-	-	-	-	-	-	-
simulation paradigm	DTS ^h	DES	DTS ⁱ	SD	-	DTS ^j	n/s	SD	MAB	H	n/s
integrating industry standards for software process description	-	-	-	-	-	-	-	-	-	^k	^l
utilizing real world tools, used in SE contexts	-	-	-	-	-	-	-	-	+	-	-
componentized for partial reuse	-	-	o	-	-	-	-	o	+	+	-
separation of process model from interaction model	-	-	-	-	-	-	-	o	+	+	-

"+" = yes/included, "-" = no/not included, "o" = partially included, simulation paradigms: "DTS"=Discrete Time System, "DES"=Discrete Event System, "SD"=System Dynamics, "MAB"=Multi Agent Based, "H"=Hybrid Simulation, "n/s"=not specified

Table 2.13: Existing DBGL Approaches In SE Education

^a Players are collaborating in teams. No competition.

^b only for administrative purposes

^c Two models provided by the AMEISE project.

^d "work-to-do, review, rework, work-completed cycle"

^e "life cycle of the project management process according to PMBOK"[153]

^f "a new, rule-based modeling language [...] that provides the required flexibility."[290]

^g The approach provides the ability to define scenarios, loadable by the simulator.

^h "operates on an incremental, time-step basis, allowing [...] actions every clock tick"[174]

ⁱ "a time-discrete simulation mechanism [...] allows for a quasi-continuous simulation"[76]

^j since it claims to be based on SimSE

^k extracted rules from CMMI

^l based on and linked with PMBOK

SE endeavor presented. OSS provides a static model, not intended to be customized by users.

The *Incredible Manager*[18, 67] presented by Dantas et al. was introduced in 2006. It is a simulation-based game that aims at providing experiential learning to project managers. “A System Dynamics model describing a software project, a simulator, and a game machine that handles user interactions and presents simulation results in a game-like fashion compose the game.”[18] Players act as project manager planning and controlling a software project within estimated limits of budget and schedule. The simulation model is configured by a set of “phase definition files” following the *system dynamics* simulation paradigm, used by many industrial simulation models. Built for training of project managers this approach has a strong focus on project management.

SimVBSE introduced by Jain and Boehm[125] in 2006 is a game built for students to better understand *value-based software engineering*[24, 29] and its underlying theory. *SimVBSE* is highly interactive and offers players to visit a set of rooms, to get tutorials, collect feedback, analyze project and organization metrics, risks and investments, to meet stakeholders, to buy information, review technology as well as candidates, and to make decisions by changing project parameters. Since this game was built solely for promoting the VBSE method, its underlying engine, or model, is neither customizable nor provides alternative processes or SE methods.

In 2007 Ye et al.[306] reported about an attempt to utilize 3D virtual worlds, *SecondLife* in this case, to learn SE topics. This attempt was twofold. Firstly, based on the *Groupthink Exercise*[83], developed at the MIT, students performed this activity by attending a session in *SecondLife*. This approach was chosen to overcome limitations of the hardware, suggested for this purpose.

Secondly, the authors developed a *Multiplayer Online Software Engineering Process (MO-SEProcess) game* based on the *SimSE* project (discussed later). The authors describe that in this game, six SE roles are provided, from which players have to choose one. Players joining the game form a software project team interact in the *SecondLife* world “through various communication means.” If a team can deliver the required product before the deadline, the team gets a score. “To get the good score, all the players need to not only work on their own part, but also collaborate with each other.”[306] Unfortunately, the authors provide no details, neither about the provided roles and what concretely players are doing on their own part nor about the collaboration between players. Since in the *SimSE* game the player takes a quite different role as project manager, the provided abridged

description does not reveal, what this game really is about. Details about educational approaches taken and the underlying simulation model are not provided..

In 2005 Shaw and Dermoudy[250] presented a simulator entitled *SimjavaSP*. They describe, the “goal of our simulator [...] is for the student, acting as the project manager, to develop a software project within the required time and budget, and of acceptable quality. This will require students to optimize the three factors of time, expenditure and quality in parallel.” This approach provides—intentionally—no tutor or help function. In author’s opinion, explicitly drawing on the SESAM project (discussed later), this design would ensure that educational objectives are achieved. In their opinion “[...] the educational success of a training environment strongly depends upon the fact that the student is not guided by the system. Rather, as in real software projects, the project manager is entirely responsible for planning, staffing, directing, and controlling. In this way, students are forced to manage the project on their own, and will therefore perceive the project outcomes as a direct result of their own decisions.”[250]

Graphical and textual feedback is provided to the players continually as the project progresses. “Throughout the project development, the student has absolute control over the developers, and can perform tasks such as hiring or firing them at any time. The student is also responsible for assigning developers to software development tasks, monitoring the project’s size, budget and time, calling progress meetings, and ensuring that the project is of high enough quality.” The game introduces stochastic events like changed requirements and quitting developers. The game is ended when the project is 100% complete or the player runs out of schedule or budget. The simulator driving the game is based on the DES paradigm and provides just one process model, that got not further described.

Starting in 1992 Ludewig et al.[76, 77, 161] have been presenting their approach titled *Software Engineering Simulation by Animated Models (SESAM)*. *SESAM* provides an interactive generic simulator, an own modeling approach including an own domain specific language, as well as a predefined simulation model, the *quality assurance (QA) model*. This QA model promotes quality assurance activities.

In a *SESAM* simulation exercise, a student takes on the role of a project manager controlling the simulated project through a textual interface. As the project manager, the student hires team members, assigns tasks to them and controls the progress and quality of the project. After the simulation finished, an analysis tool is provided to enable students to analyze overall project’s results as well as results of their own decisions. After the conduction of a case study and

a controlled experiment. An approach to integrate the simulation exercises into a course setting was developed.[161]

The authors summarized “In our current and future work, we try to find an appropriate mixture of lessons, playing, reflecting on the results, and playing again. Our first experiments have shown that just playing is not enough, because the students are not able to understand the reasons why they have failed. Their behaviour while playing is very similar to their behaviour when they actually develop software: They do not really trust in the lessons they (should) have learned, so they are controlled by their emotions.”[77]

Based on the work of SESAM Mittermeir et al.[34, 35, 167] developed *A Media Education Initiative for Software Engineering (AMEISE)*. This project aimed to extend the core ideas of SESAM and “at extending the spectrum of educational situations where this project management adventure game can be played.” AMEISE replaced the “text-based, pseudo-natural language user interface. [...] by an interface the current PC-generation of students might be more used to” and added support for players, by integrating “friendly peers”, which give pro-active advice in situations of severe mistakes, and “consultants”, which might be actively invoked by the student in situations, where additional help is needed. This was accomplished by wrapping SESAM, with all its special requirements to its runtime environment, and enhancing the architecture with components needed. The authors note that “interaction with the system became markedly easier and students had to worry less about intricacies of the system but could better concentrate on the complexity of the actual project management task. It is up to the instructors though to make sure that simulations do not degenerate to a click-and-try venture. An important educational aim is to make students aware that managing a project of this size requires active planning and constant monitoring. Acting too short-sighted just on the feedback obtained from the system and its virtual developers will undoubtedly lead to missing the objective function to be reached by this simulated development.”[167] The QA simulation model of the SESAM project was complemented by a tutorial-like mini model and a maintenance model focusing on maintenance tasks.

AMEISE has been utilized with success for over a decade by different institutions and different target groups.[34, 35]

In 2004 Navarro et al.[174, 175, 180–182] presented *SimSE*, which likely is the most advanced and most comprehensive approach utilizing DGBL in SE education so far. SimSE provides a number of prepared software process models. As the period of years suggests, this approach has been developed much beyond its initial version presented in 2004. In SimSE, a player controls a virtual project team

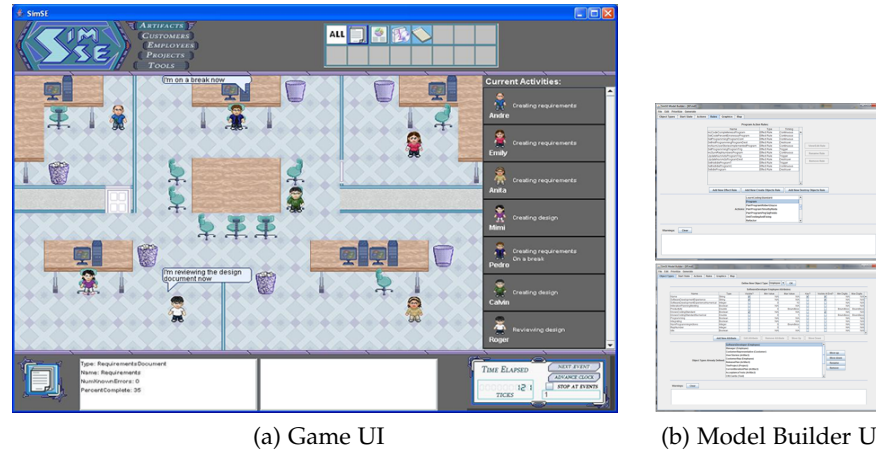


Figure 2.7: SimSE[174–176, 180–182]

and guides this one through a software project following the given software process. This game provides a graphical UI showing the team at work as well as artifacts growing as result of assigned activities. After playing the game, a report is presented to enable analysis of the results. SimSE provides an own modeling approach using a graphical model builder. The user interfaces for the game and the model builder are shown in figure 2.7. Since *SimSE* has been providing a very popular environment over the years, the environment was used in a number of studies, e.g. to build new simulation model concerned with GSD[307] or to be used in a non-game context to verify and validate process definitions[110].

Software process models provided by *SimSE* include the *Waterfall* model, an *Incremental* model, an *Inspection* model, a *Rapid Prototyping* model, the *Rational Unified Process* model and the *Extreme Programming (XP)* model.[257] Navarro[174] states “Although the model builder removes the inherent difficulties of a programming language (e.g., syntax), we recognize that the difficulty of collecting software engineering phenomena and rules and translating these into SimSE actions and rules still remains. [...] It is important to note that use of the model builder also does not guarantee the model is a ‘good’ model. Rather, a strongly iterative development cycle is required. In our experience so far, building a model involves a significant amount of time aside from the initial construction of the model in which the model is repeatedly played and refined in order to ensure that the desired lessons and effects are illustrated, as well as to achieve the desired balance between educational effectiveness and realism [...].” She acknowledges “there are still some weaknesses to our approach, which lie mainly in the fact that it lacks many common programming language constructs, such as if-else statements, explicit data structures, loops, and predicates. This makes it necessary at times to use some non-intuitive, roundabout techniques to get the desired effect.” *SimSE* has been thoroughly analyzed and evaluated in a number of

replicated and multi-site experiments since its initial introduction[178, 180–182].

Caulfield et al.[49] presented *Simsoft* in 2011. *Simsoft* is focused “specifically at human resources aspects of the build phase of a software engineering project”[49] Different from other approaches already described, *Simsoft* physically comes in two pieces. The first one is an “A0-sized printed game board around which the players gather to discuss the current state of the project and to consider their next move. The board shows the flow of the game while plastic counters are used to represent the staff of the project. Poker chips represent the team’s budget, with which they can purchase more staff, and from which certain game events may draw or reimburse amounts depending on decisions made during the course of the game.” The second piece is a simple Java-base dashboard providing the current and historical state of the project and enabling players to adjust project settings, e.g. by recruiting new virtual team members.

Players of *Simsoft* are organized into teams of two or three or more and provided with a scenario describing requirements for a small software project. The team acts as project manager and controls a virtual software project from start to final delivery. With a focus on quality assurance, the project follows a “work-to-do, review, rework, work-completed cycle.” Players may decide to hire staff, starting work with delay, with different levels of experience and at different prices. The underlying simulation model driving the dashboard is based on System Dynamics models and virtual time advances as requested by the team. Depending on the levels of experience of the virtual team, more or less rework may be needed. The objective of the game is to deliver the software in the schedule and budget.

Peixoto et al.[201, 202, 204] introduced *SPIAL (Software Process Improvement Animated Learning Environment)*, a single player game based on a framework[202] to develop SE simulation games. This gameplay of this serious game is combining a waterfall process model with Software Process Improvement (SPI). “The player is given a process improvement task and he/she can interact with other stakeholders (high level management, project manager, team member, consultant, or customer) represented as nonplayer characters [...]. In order to complete the task, the player can make investments for improving specific process areas of a software development project (which can be considered a pilot project). A good investment strategy will result in improvement of process areas and a bigger budget for further investments. *SPIAL* incorporates some of the concepts defined in CMMI-DEV version 1.3.”[201] The Capability Maturity Model Integration for Development (CMMI-DEV) was defined by the *Software Engineering Institute of the Carnegie Mellon University* and “provides a

comprehensive integrated set of guidelines for developing products and services.”[258]

Peixoto et al.[201] state, “we were interested in rules that can be applied in a simple waterfall development project, covering some concepts of CMMI. In most of the cases, either these rules are too generic or they are from other domains (e.g. agile software development). We restricted the set of the rules to be applied in this simulation game (57 of 123 rules). We mapped each rule to a specific process area of the model (e.g. Requirements development) and the corresponding measurements that they can impact (e.g. defects and productivity).” The game is reported to use a hybrid simulation approach combining discrete event and continuous approaches.

The *Project Management Game 2D (PMG-2D)*, presented by Lino et al.[153] is “a serious educational game, which intends to help inexperienced project managers to be trained, considering the management of costs, time, risk and human resources.” This game is oriented towards the Project Management Body of Knowledge (PMBOK)[219] provided by the *Project Management Institute (PMI)* and follows a rather non-SE specific “life cycle of the project management process according to PMBOK: initiation, planning, execution, monitoring/-control and finishing.”[153] The underlying model of the game describes non-player characters in with a rich set of personality factors, including “stress, food, physical, health, relationship (with friends and family), motivation, adaptation, interest, experience, training, co-operation (with company employees).”[153] These attributes are influencing their behavior. “The main goal of the player is to create a roadmap for the project.”[153] In the game, a player has to identify project stakeholders, build the team as demanded and allocate it to respective tasks. To support the player, the game shows alerts at mistakes and provides suggestions. Chapter and section of the PMBOK are provided to the player to invite for further reading about specific topics.

Nassal[172] presented his *project management game* in 2015. What makes this work exceptional is the use of a real world tool to drive the game. Based on the impression that game-based approaches in SE education lack the utilization of real tools used in everyday software development projects[135, 172, 213] this game is utilizing *Microsoft Project* and add-in components, developed for this approach, as a user interface to drive a generic simulation model. The generic simulation model[171] is based on Multi-Agent-Based Simulation (MABS) and not bound to any particular software process model. It gets parametrized and customized by loadable scenarios. This approach ensures high flexibility and enables the reusability of the model in other contexts, e.g. in combination with other interaction models.

Players in this game control a virtual software project and follow a given software process. Documentation describing the process is integrated into the game. By planning the project flow in Microsoft Project, e.g. by planning the sequence, dependencies, and effort, the virtual team accepts tasks generated from the planned activities. Tasks are assigned to specific project members, each modeled with a set of individual attributes and attitudes, via the resource assignment mechanisms of the project planning tool. Virtual team members use these resource assignments as a recommendation and fulfill their tasks depending on their defined attitudes. Feedback is provided via a project status report available throughout the game. The deterministic simulation can be restarted to enable learning from failures, e.g. to correct a underutilization of workload for a given virtual project member. A player is basically planning the project and replanning if this should seem to be necessary. The player stops the game when it seems appropriate. A provided score enables comparison of results after the gameplay.

The approaches described above are summarized in table 2.13 on page 72.

2.5.2 *Evaluation*

Wangenheim and Shull[301], authors of a meta-study, summarized findings of utilizations of DGBL in SE education. Most studies in their review used a non-experimental research design without control groups or other comparison and conducted no pre-test to assess student's abilities.

The authots mention that most articles did not clearly provide descriptions of learning objectives and environments in which students used the game. One-third of the studies could not ascertain evidence of improvement in learning. "Overall, however, most studies showed at least a minor impact from using a game. [...] Several studies reported mixed results (games facilitated some learning outcomes but not others). Yet it seems that, in general, game-based learning appears to have more impact on lower cognitive levels, reinforcing knowledge learned earlier. It seems inadequate for teaching new knowledge." [301]

As many studies reported, students prefer game-based approaches over traditional instructional methods and acknowledge an increased appreciation of SE practices' importance. The degree of enjoyability showed variations and studies noted that players might lose interest in games they see as too complex or too long.

Wangenheim and Shull recommend that the way we evaluate games' learning impact should become more formal to increase the level of evidence.

Jiang et al.[126], authors of a recent SLR, provide a list of recommendations to follow at implementing simulation- and game-based learning activities.

2.5.3 Conclusions

The educational approaches taken, vary in the described attempts to utilize DGBL in SE education.

*Simsoft*_[49] is explicitly oriented towards *Problem-Based Learning*. All approaches represent to more or less extent constructivist learning approaches by providing kinds of *experiential learning*, *active learning*, *learning by doing*, *situated learning* and *discovery learning*.

The amount of *scaffolding* provided differs considerably. *SimjavaSP*_[250] explicitly does “not guiding” the learner, which represents kind of minimally guided instruction (tending towards unguided instruction). Criticisms on that approaches are described in section 2.2 on page 19. Considering Vygotsky’s *Zone of Proximal Development* (ZPD), Sweller’s *Cognitive Load Theory* (CLT) and Reigeluth’s *Elaboration Theory* (cf. section 2.2 on page 19) this might not contribute to ideal learning experiences.

Concepts of *Social Constructivism* (cf. section 2.2.3 on page 24) seem clearly to be underemployed in existing approaches.

Players of *Simsoft* appreciated to work in teams, to share opinions and learn from more experienced teammates.

The extent to what learners collaborated in *MO-SEProcess* is not clearly described.

All other approaches are provided as single player games. In single player games, collaboration and motivating competition between isolated participants are not promoted. The architectures of the game environments, except those of the *AMEISE* project, do not enable a lecturer to get a quick summary of the performance of all players in a course. Such an overview could facilitate quick, supportive individual or group interactions in the case of occurring misconceptions, as well as support debriefing activities.

The usage of real world tools, as implemented in Nassal’s *Project Management Game*_[172], enables the beneficial development of skills in the knowledge domain *and* the utilized tool. On the other hand it makes the approach dependent on the tool. If the chosen tool does not represent the acknowledged mindset or essential principles of a given SE method, the approach may be rejected for just that reason. For instance, it might be less accepted to play a *Scrum*_[140, 141, 146] or *Kanban*_[8] method based on *Microsoft Project*, since these agile methods do not advocate elaborated long-term plans and are based

on pull-mechanisms.

Almost all authors recognize that their games alone are not sufficient learning vehicles and hence have to be complemented by other educational approaches. Game-based learning activities have to be integrated in the context as a whole in order to address intended learning objectives.

Most approaches offer only one fixed model of a software process or similar concepts. Taken the existing diversity of software processes into account, model customization capabilities seem crucial in today's software engineering education.

To transfer knowledge gained in one specific process model to new upcoming challenges requiring different practices is an exercise left to students. None of the existing approaches facilitates a perspective towards the use of flexible SE practices instead of software processes yet (cf. section 2.6.1).

The creation and customizing of simulated models require considerable initial training efforts, which are not directly of use outside those specific game environments. New software process models often have to be built from scratch without the opportunity to reuse existing ones.

Simulation models, except the one used in Nassal's *Project Management Game*, tightly integrate simulator and game interface. The reuse of the simulator in another context, e.g. using other interaction approaches, is impeded.

Most simulation based approaches lack integrated documentation of the process or method to follow. Without such descriptions, players are strongly dependent on prior knowledge to accomplish the game—given same vocab with same semantics is used. By assigning activities, represented by just a word or short word group, concepts and semantics may keep unclear.

Most simulation based games, enabling the learning of whole processes/methods, promote a focus on workload optimization of team members with specific skills. Given the increasing recognition of agile methods based on pull-approaches, where team members choose appropriate work packages depending on their own abilities, this seems not to be an essential task in *all* SE contexts anymore. Unfortunately, it occupies a lot of learners attention—that could be focused on other essential problem-solving issues otherwise.

While all of the introduced approaches provide an opportunity to familiarize with the field of software processes and their manage-

ment, each with different focus and approach, they were not explicitly built to directly support students in the course or capstone project work that is demanded by curriculum guidelines (cf. section 2.3.3 on page 46).

If SE students should acquire an SE mindset, truly appreciating the guidance and support of utilized methods and practices, they should be provided with experiences facilitating such perceptions. In author's opinion DGBL in SE education could, and should, provide such a direct support of real project work.

2.6 SOFTWARE PROCESSES, METHODS, AND PRACTICES

The SWEBOK[112] states “Software processes³¹ are specified for a number of reasons: to facilitate human understanding, communication, and coordination; to aid management of software projects; to measure and improve the quality of software products in an efficient manner; to support process improvement; and to provide a basis for automated support of process execution.”

According to the SWEBOK “a software process is a set of inter-related activities and tasks that transform input work products into output work products. At minimum, the description of a software process includes required inputs, transforming work activities, and outputs generated. [...] a software process may also include its entry and exit criteria and decomposition of the work activities into tasks, which are the smallest units of work [...] Complete definition of a software process may also include the roles and competencies, IT support, software engineering techniques and tools, and work environment needed to perform the process, as well as the approaches and measures (Key Performance Indicators) used to determine the efficiency and effectiveness of performing the process.”

SWEBOK describes *software life cycle models* as a category of the *software process*. “A software development life cycle (SDLC) includes the software processes used to specify and transform software requirements into a deliverable software product. A software product life cycle (SPLC) includes a software development life cycle plus additional software processes that provide for deployment, maintenance, support, evolution, retirement, and all other inception-to-retirement processes for a software product, including the software configuration management and software quality assurance processes that are applied throughout a software product life cycle. A software product life cycle may include multiple software development life cycles for evolving and enhancing the software.”[112]

Software Engineering Methods are described by SWEBOK in an extra Knowledge Area (KA) with the same title. “Software engineering methods provide an organized and systematic approach to developing software for a target computer. There are numerous methods from which to choose, and it is important for the software engineer to choose an appropriate method or methods for the software development task at hand; this choice can have a dramatic effect on the success of the software project. Use of these software engineering methods coupled with people of the right skill set and tools enable the software engineers to visualize the details of the software and ultimately transform the representation into a working set of

³¹ The term *software engineering process* is often referred to as *software process*. For instance, the SWEBOK[112] states: “For readability, ‘software engineering process’ will be referred to as ‘software process’ [...]” Another interchangeably used term is *software development process*.

code and data.”[112] *Agile Methods* form a sub-area of this KA, along with *heuristic methods*, *formal methods* and *prototyping methods*. “Agile methods are considered lightweight methods in that they are characterized by short, iterative development cycles, self-organizing teams, simpler designs, code refactoring, test-driven development, frequent customer involvement, and an emphasis on creating a demonstrable working product with each development cycle.”[112]

From SWEBOK’s perspective the knowledge area *Software Process* is closely related to the knowledge areas *Software Engineering Management*, concerned with tailoring, adapting and implementing software processes for specific software projects, and *Software Engineering Models and Methods*, supporting a systematic approach to software development and modification.

However, terms in the field of software processes are not used without ambiguities. Laplante[151] for instance defines *software process* shortly: “A software process is a model that describes an approach to the production and evolution of software. Software process models are frequently called ‘life-cycle’ models, and the terms are interchangeable.” This definition does not differentiate between a *process*, a *process model*, and *life cycle models*.

Sommerville[260] provides a somewhat more detailed definition. He defines a *software process* as “[...] a sequence of activities that leads to the production of a software product”[260]. He mentions “[...] four fundamental activities that are common to all software processes. These activities are:

1. Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.
2. Software development, where the software is designed and programmed.
3. Software validation, where the software is checked to ensure that it is what the customer requires.
4. Software evolution, where the software is modified to reflect changing customer and market requirements.

Different types of systems need different development processes.”[260]

From another perspective the terms *software (engineering) process* and *software (engineering) method* are used interchangeably: “From henceforth, we will use the term processes and methods interchangeably, preferring the word ‘method’ over ‘process’.”[183]

Such ambiguous use of vocab with—sometimes more and sometimes less slightly—varying semantics is not unusual in this field. Actually it is rather ubiquitous. A reader has to carefully acknowledge the context to grasp the current meaning of terms used. In the context of this thesis, a pragmatic approach is taken. Where a precise differentiation between *software process*, *software lifecycle*, *software (engineering) method* is needed, it will be mentioned. Otherwise, the term *software process* and *SE method* are used interchangeably, preferring the common use in the respective context, e.g. of authors or publications referred to. Depending on the context, a *software process* or *SE method* may represent a complete *life cycle* (SDLC or SPLC), but it does not have to necessarily.

“Software processes are complex and, like all intellectual and creative processes, rely on people making decisions and judgments. There is no ideal process and most organizations have developed their own software development processes. Processes have evolved to take advantage of the capabilities of the people in an organization and the specific characteristics of the systems that are being developed.”[260]

As a result, we find a considerable number of SE methods today. Kennaley[142] lists 40 SE methods with their origins in his book.

Péraire and Zapata[208] mention that out of the 40 SE methods only “a minority (about 15%) originated from Academia or was significantly influenced by Academia”, including none from the agile or lean lineage. According to them, 32 methods emerged in the last 20 years, resulting in 1.6 methods per year. From an SE education perspective, this is a challenging fact.

Taking into account, that almost each project team does, or should do, some customization of a chosen SE method, the amount of existing SE methods is tremendous in practice.

“Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes [...] planning is incremental and it is easier to change the process to reflect changing customer requirements. [...] each approach is suitable for different types of software.”[260]

The first one, *plan-driven processes*, were used for decades by large teams constructing software for extended periods of time. As Sommerville states “These plan-driven approaches involve a significant overhead in planning, designing, and documenting the system. This overhead is justified when the work of multiple development teams has to be coordinated, when the system is a critical system, and when

many different people will be involved in maintaining the software over its lifetime.”[260]

The latter, *agile processes*, or *methods*, have been arisen from the awareness that this significant overhead was less justified at developing smaller business systems in rapidly changing environments. To derive a complete set of aligned and stable requirements before starting development activities became practically impossible in fast changing business contexts.[260] Agile methods have become highly prevalent since the *Agile Manifesto*[19] published in 2001. According to a study of *VersionOne*[297], *Scrum* (58%) and *Scrum/XP hybrid* (10%) are dominating methods in the agile field.

Proponents of both *plan-driven* and *agile* approaches passionately defended their position in an arisen “*method war*”³² on an “*emotionally complicated topic*.”³³ In an attempt to “clarify the perplexity about the roles of discipline, agility, and process in software development” Boehm and Turner[30, 32] compared and contrasted traditional plan-driven and agile approaches, their characteristics, strengths, weaknesses, and misconceptions. Table 2.14 on the facing page summarizes their findings on common basic assumptions of both approaches. Boehm and Turner identified five critical factors, or dimensions, to determine the suitability of an agile or plan-driven method in a given context: *project’s size*, *criticality*, *dynamism*, *personnel*, and *culture factors*. The first two factors imply rather plan-driven approaches at raising values and agile methods at lower levels. *Dynamism* implies agile methods in highly dynamic environments and tends to plan-driven at highly stable environments.

The SWEBOK mentions[112] “It must be emphasized that there is no best software process or set of software processes. Software processes must be selected, adapted, and applied as appropriate for each project and each organizational context. No ideal process, or set of processes, exists. [...] there will always be a place for heavyweight, plan-based software engineering methods as well as places where agile methods shine.” Sommerville concludes “Generally, you need to find a balance between plan-driven and agile processes.”[260]

Wang et al.[300] state “In recent years however the agile community has started to look toward lean software development approaches, in addition to agile methods such as XP and Scrum.” The adoption of *lean thinking* in agile methods resulted in *lean methods* like *Kanban* and

32 Grady Booch in his foreword to “Balancing Agility and Discipline: A Guide for the Perplexed”[30]

33 Alistair Cockburn in his foreword to “Balancing Agility and Discipline: A Guide for the Perplexed”[30]

Characteristics	Agile	Plan-Driven
<i>Application</i>		
Primary Goals	Rapid value; responding to change	Predictability, stability, high assurance
Size	Smaller teams and projects	Larger teams and projects
Environment	Turbulent; high change; project-focused	Stable low-change; organization focused; low-change;
<i>Management</i>		
Customer Relations	Dedicated on-site customers; focused on prioritized increments	As-needed customer interactions; focused on contract provisions
Planning and Control	Internalized plans; qualitative control	Documented plans, quantitative control
Communication	Tacit interpersonal knowledge	Explicit documented knowledge
<i>Technical</i>		
Requirements	Prioritized informal stories and test cases; undergoing unforeseeable change	Formalized project, capability, interface, quality, foreseeable evolution requirements
Development	Simple design; short increments; refactoring assumed inexpensive	Extensive design; longer increments; refactoring assumed expensive
Testing	Executable test cases define requirements	Documented test plans and procedures
<i>Personnel</i>		
Customers	Dedicated, collocated CRACK* performers	CRACK* performers, not always collocated
Developers	more experts needed for the whole endeavor	more experts needed early; less experts later
Culture	Comfort and empowerment via many degrees of freedom (thriving on chaos)	Comfort and empowerment via framework of policies and procedures (thriving on order)
Examples	eXtreme Programming (XP), Scrum, Dynamic Systems Development Method (DSDM), Crystal Methods, Feature-Driven Development (FDD)	Waterfall, Rational Unified Process (RUP), Spiral, EnterpriseUP
* Collaborative, Representative, Authorized, Committed, Knowledgeable		

Table 2.14: Agile and Plan-Driven Method Home Grounds [adapted from 30]

terms like *leagile*, combining *lean* and *agile*, or *Scrumban*³⁴, a combination of *Scrum* and *Kanban*. Lean approaches are “believed to be especially suited for maintenance projects or projects with frequent and unexpected user stories or programming errors. In such cases, the time-boxed sprints of the Scrum model are of no appreciable use, but Scrum’s daily meetings and other practices can be applied, depending on the team and the situation at hand.”[300]

The core objective of a Kanban system is to limit the *work in progress* (WIP) to “keep the process flowing at an even but continuous rate.” Work is only considered to be done (*just-in-time*) if it was *pulled*, demanded by a downstream process. Lean software development promotes a set of principles and practices that are partially overlapping with agile methods. The *identification and elimination of waste* as well as *continuous improvement* (*kaizen*), focused on optimizing the whole, are primary guiding principles. “In the domain of software development, the types of waste can be interpreted as: extra features, waiting, task switching, extra processes, partially done work, movement, defects and unused employee creativity.”[300]

“Agile methods also advocate continuous improvement, but do not answer how it can be implemented. Lean approaches, instead, offer specific directions (eliminating waste in software development processes, focusing on flow, etc.) and specific practices (Kanban, value stream mapping, root cause analysis, etc.) to improve agile processes continuously.”[300]

Agile methods are considered to be designed focused on the needs of software development and its management but not to scale well at a programme, product, or organization level without change. Proponents see fundamentals for such change in lean principles since “lean principles can be applied to any scope, from the specific practice of developing software to the entire enterprise where software development is just one small part.”[300]

2.6.1 Towards SE Practices Instead Of Software Processes

“We need to consider whether we are educating children for their futures or our pasts.”
—Geoff Southworth
(2002)

In reaction to the high number of existing SE methods, critics[121–123, 142, 162] question their *raison d’être*.

In his book published in 2010 Kennaley[142, 162] calls to “end the iterative method wars.” He assumes, that each approach of SE methods has something to offer in its particular context, and argues, that instead of a wasteful competition of SE methods their inherent practices should be integrated. Kennaley complains about a lack of trust

³⁴ When asked, in an annually report, what agile/lean methodology is followed most closely, 7% of respondents answered to practice *Scrumban*, 5% to practice *Kanban*, and 2% to practice *Lean Development*. (compared to 58% respondents practicing *Scrum* and 10% practicing *Scrum /XP hybrid*)[297]

between different communities resulting in a *us-vs-them* attitude. To demonstrate existing common ground, he presents a domain model of practice patterns. By associating practices from the *agile*, the *lean*, and the *Unified Process* communities, he demonstrates that one can find community specific innovations as well as synonyms and re-brandings of existing practices. With rooting his considerations in systems theory, Kennaley suggests a new *SDLC 3.0*, after waterfall-like development being version 1.0 and iterative/incremental development being version 2.0, combining approaches from all camps in a *complex adaptive system of patterns*.

In a series of articles, published in 2007, Jacobson et al. call “*Enough of Processes: Let’s Do Practices*”[121–123]. They acknowledge that all modern software processes “try to help project teams conduct their work.”[122]. They state that “While there are some important differences between them, the commonalities are far greater—and understandably, since the end goal of them all is to produce working software quickly and effectively. Thus, it doesn’t matter which process you adopt as long as it is adaptable, extensible, and capable of absorbing good ideas, even if they arise from other processes.”[122]

Instead of ongoing search for the *Silver Bullet*[44] the SE community should strive for flexibility. In order to achieve such flexibility, they argue “[...] things need to change. The focus needs to shift from the definition of complete processes to the capture of reusable practices. Teams should be able to mix-and-match practices and ideas from many different sources to create effective ways of working.”[122] Jacobson et al. list six serious problems of software processes interfering effective and efficient work of teams demanded to follow them. The six identified problems are summarized in table 2.15.

To overcome those problems, Jacobson et al. proposed to utilize practices instead of processes, hiding them inside. “A ‘practice’ provides a way to systematically and verifiably address a particular aspect of a project. [...] A practice has a clear beginning and end, and tells a complete story in usable chunks. A practice includes its own verification, providing it with a clear goal and a way of measuring its success in achieving that goal.”[121] According to the authors, the problems mentioned above could be solved with such qualities: “practices can be developed, learned, and adopted separately, and they can be used in conjunction with other practices to create easily understood and coherent ways-of-working.”[121] From this perspective, software processes would represent (just) a collection of “typically tangled and tightly coupled practices” drawing on existing practices and acting as a starting point or goal for projects. By choosing only the practices needed, a team or organization would not need to down-cut a software process to a digestible format anymore. The au-

The Problem of...	(Shortened) Description
<i>Denied Commonality</i>	"each process has a few interesting gems, but they are embedded in a larger package of commonalities. [...] Because everything in every new process looks new, it can be hard to really compare processes. It is even harder to mix and match—extract the gems and combine them with the gems from other processes."
<i>Completeness</i>	"each process definition—large or small—wants to describe a complete process. [...], typically in a tightly coupled and homogeneous fashion. [...] By striving for completeness, the processes end up as brittle, all-or-nothing propositions. This makes it hard for teams to adopt just the good parts of the process and identify truly new ideas and techniques. [...] This leads to the need for organizations, practitioners, and methodologists to start trimming to get a lighter, more-focused process."
<i>Adopting a Complete Process</i>	"each software team has its own way of working (explicit or tacit) [...] there are always good practices that they will want to continue using. Other areas of the process will be weaker and lead to the desire for change. [...] branded processes do not address this reality and require the team to change everything just to get the few new things that they want."
<i>Out-of-Sync Process</i>	"What a team actually does never matches the adopted process. Teams improvise and invent more effective ways of doing things; they find they need to work out solutions to problems nobody thought of when the process was selected, and they never keep the process descriptions up to date. [...] it becomes difficult to spread [...] success to other teams [...] It makes it difficult to plan, estimate, monitor, and control projects. Process and quality assurance becomes less effective and more expensive."
<i>Acquiring Knowledge</i>	"people want to apply processes, not read about them [...] they want different levels of detail at different times. [...] actively engaged in developing software they want succinct, focused, unambiguous guidance that will immediately help them undertake their work, and not long explanations, anecdotes, and academic treatises that justify the techniques they are trying to apply."
<i>Stupid Processes</i>	"the process [...] as a passive knowledgebase [...] doesn't interact [...] and offer [...] appropriate and timely advice [...] expects you to know exactly what you are doing and exactly what useful information it contains and exactly where it has hidden it. [...] Without the ability to take an active role in helping people to develop software, [...] teams will continue to struggle to realize its benefits."

Table 2.15: Problems Caused by Current Generation of Software Processes[122]

thors propose a *kernel* and *cards* to make practices tangible.[123]

With those ideas Jacobson et al. laid the foundation for an initiative and an OMG specification that should be published in its first version seven years later: *SEMAT Essence*[187, 188] gets described in section 2.7.

2.6.2 Meta-Models for Standardized Process Description

A number of industry initiatives have been striving for ways to standardize the description of software processes. These initiatives developed a number of meta-models, process frameworks, and languages to that purpose, with being *Software and Systems Process Engineering Meta-Model (SPEM)*[186], *Software Engineering-Metamodel for Development Methodologies (ISO24744)*[114]—and recently added—*SEMAT Essence*[187, 188] the most prominent international³⁵ ones.

“Software processes
are software too.”
—Leon J. Osterweil
(1987)

Kuhrmann et al.[149] provide an overview and a timeline with relevant software process meta-models, their origins, evolutions, and extensions. They state that they “could see, for example, that [...] ISO 24744, although being proclaimed as an industry standard, have to the best of our knowledge no practical relevance as they remained at the level of a complex metamodel specification. They neither have an observable active community in the sense of developing new releases or tools to support process engineers and users, nor are any documented experiences available that would indicate to a certain dissemination.”

Software Process and Systems Engineering Meta-Model (SPEM) is defined as a meta-model as well as a UML 2 profile. Ruiz et al.[233] describe the “main objective of SPEM, like any other language for processes, is to provide the building blocks required to represent software processes in a standard way.” A wide range of processes is modeled utilizing SPEM. These processes range from the iterative *Unified Process* over agile methods like *Scrum* and *eXtreme Programming (XP)*, *Agile Business Rules Development (ABRD)* methodology, *Dynamic Systems Development Method (DSDM)*, “to specific methodologies such as those for the development of health systems.”[233] With its definition as meta-model and UML 2.0 profile, several tools support SPEM. Those include *Unified Modeling Language (UML)*[190] tools as well as the open source software *Eclipse Process Framework (EPF)*³⁶ with the

³⁵ Kuhrmann et al.[149] also mention the German V-Modell XT meta-model, but mention “It remains, however, a purely German standard with little international attention.” The authors consider the OPEN process framework to be a predecessor of ISO24744[221].

³⁶ <https://eclipse.org/epf/>, The EPF is built on the Unified Method Architecture (UMA), an evolution of SPEM 1.1.[99]

*EPF Composer*³⁷ and commercial tools like the *Rational Method Composer (RMC)*³⁸. SPEM is explicitly “targeted at process engineers, [...] responsible for maintaining and implementing processes for their development organizations[...].”[186] To enact processes defined in SPEM, they are typically mapped to project planning tools or workflow engines, since “the language architecture itself lacks built-in enactment capabilities.”[42] Ruiz et al.[233] mention a number of extensions proposed to enable process enactment.

The most recent attempt to provide a standardized SE method description is *Essence—Kernel and Language for Software Engineering Methods*[187, 188], Object Management Group (OMG) specification provided by the SEMAT[245] initiative. This approach differs considerably from approaches existing before:

1. by its orientation towards not only process engineers but “*practitioners as well as method engineers*”[188],
2. by providing a *kernel* of foundational concepts inherent to all SE endeavors, and
3. by providing *dynamic semantics* facilitating the *enactment* of SE methods.

With this characteristics, this approach significantly supports the objectives of the *Integrated Approach* introduced in this thesis. *SEMAT* and *Essence* get introduced in detail in the following section 2.7 on page 95.

Table 2.16 on the facing page summarizes characteristics of SPEM, ISO24744, and SEMAT Essence. A mapping of concepts and language as well as meta-model elements between SEMAT Essence and both SPEM as well as ISO24744 is provided by the Essence specification[188]. Elvesæter et al.[42] provide a detailed mapping of SPEM and Essence concepts.

Taking the eight years since its release, as well as the *EPF*, which is still based on UMA and not yet SPEM 2.0, into account, the commitment to further development of SPEM seems rather questionable.

2.6.3 Conclusions

Besides a large number of software processes and methods, covering a wide range of perspectives, we find strong arguments for an orientation towards SE practices instead of SE processes today. Such an orientation is supporting SE practitioners as well as SE education.

³⁷ https://eclipse.org/epf/downloads/tool/tool_downloads.php

³⁸ <http://www.ibm.com/software/products/en/rmc>

	SPEM	ISO/IEC 24744:2014	SEMAT Essence
full name	Software & Systems Process Engineering Meta-Model Specification	Software engineering—Metamodel for development methodologies	Essence – Kernel and Language for Software Engineering Methods
published first version	2001[149]	02/2007[113]	11/2014[187]
published last version	04/2008[186]	11/2014[114]	12/2015[188]
current version	2.0	“2nd edition”	1.1
design goals / target group(s)	“[...] define software and systems development processes and their components. [...] targeted at process engineers, project leads, project and program managers who are responsible for maintaining and implementing processes for their development organizations or individual projects.”[186]	“[...] define methodologies in information-based domains, i.e. areas characterized by their intensive reliance on information management and processing, such as software, business or systems engineering.”[114]	“designed to support practitioners as well as method engineers.”[188]
authoring tool support (OSS or cost free / commercial)	YES ^a / YES	NO / YES	YES / YES
price of specification document	free	CHF 198	free
available practice library	YES ^b		YES ^c
standardization organization	OMG	ISO	OMG
operational semantics to support enactment	NO	NO	YES
provides language/meta-model	YES	YES	YES
provides kernel of essential concepts	NO	NO	YES
supports method enactment	NO	NO	YES ^d

Table 2.16: Comparison of Industry Standards for Standardized SE Process Description

^a based on Eclipse Process Framework (EPF), based on Unified Method Architecture (UMA)—which again is based on SPEM 1.1[100]

^b based on EPF, comprehensive

^c still small but growing

^d via *dynamic semantics* defined in the Essence specification[187, 188]

Industry efforts to standardize process description provide support at overcoming a large number of formats and vocabulary used in the field. Primarily those efforts were targeted at process engineers supporting them to produce software process descriptions. With SEMAT Essence a new standard emerges that is designed

- to support SE practitioners in their everyday work as well as method engineers,
- to support method enactment to provide not just a method description but a dynamic foundation for steering an SE endeavor, and
- to provide a compact process- and practice-independent kernel of essential elements inherent in every SE endeavor.

A future-oriented approach to support SE education in the area of SE methods should take advantage of such efforts and provide support for the wide range of plan-driven, iterative, agile, and lean methods prevalent in the field.

2.7 SEMAT ESSENCE—A STANDARD FOR SOFTWARE ENGINEERING METHODS

A Call for Action[116] in 2009, pinpointing “paramount concerns and issues that challenge the field of software engineering,”[137] started the SEMAT initiative. Initiated by a remarkable number of proven experts in the field, this initiative “was very soon supported by thousands of individuals around the world, a dozen well-known companies, and about equally many academic institutions.”[137]

The *Call for Action*[116] stressed that in the field of SE there is a “lack of a sound, widely accepted theoretical basis” and a “huge number of methods and method variants, with differences little understood and artificially magnified.” It mentions that there is a “lack of credible experimental evaluation and validation” as well as a “split between industry practice and academic research.”

A vision statement[117], published in January 2010, laid out a vision, scope, goals, principles, and milestones for the start of the initiative. Following this vision statement, two major goals were focused by SEMAT[118]:

1. Finding a kernel of widely agreed-on elements.
2. Defining a solid theoretical basis for software engineering.

Jacobson et al.[118] state “To a large extent these two tasks are independent of each other. Finding the kernel and its elements is a pragmatic exercise requiring people with long experience in software development and knowledge of many of the existing methods. Defining the theoretical basis requires academic research and may take many years to reach a successful outcome.”

Progress is made at efforts focusing the second goal. A series of workshops “*General Theories of Software Engineering (GTSE)*”[106, 127–129, 222] held in conjunction with the *International Conference on Software Engineering (ICSE)* contributed to this endeavor.

As Johnson et al.[215] describe, “there exist general theories of software engineering, but they are subjective. [...] In addition to personal general theories, the software engineering discipline also features a multitude of well-known specific theories.” They argue that “small theories are disconnected from each other. They form no coherent knowledge structure. In fact, based on this mass of principles, an infinite number of contradictory statements can be derived. [...] Under these circumstances, it is no wonder that every software engineer seems to have their own general theory. [...] This inhibits the development of a cumulative body of knowledge. With a common theory, however, the scope of the field narrows, as joint investigations are concentrated to the topics designated by the theory.” The authors emphasize the need to discuss and debate a general theory as a “whole

scientific community,” as this “is arguably more likely to reflect the real world than the untested, many times unquestioned, idiosyncratic mix of ideas harbored in the mind of any given community member.”

Contributions made to these efforts include suggestions to the question how a general theory should be created[28, 215], which characteristics it should provide[28, 209], and multiple suggestions of aspects or models to include in such a general theory[150, 183, 197, 209].

As a result of efforts focusing the first goal, the “*Essence—Kernel and Language for Software Engineering Method*” specification was published by the OMG in November 2014[187] and an updated version[188] in December 2015. *Essence* gets described shortly in the following sections.

2.7.1 *Kernel and Language for Software Engineering Methods (Essence)*

Figure 2.8 shows the structural composition of *Essence*.

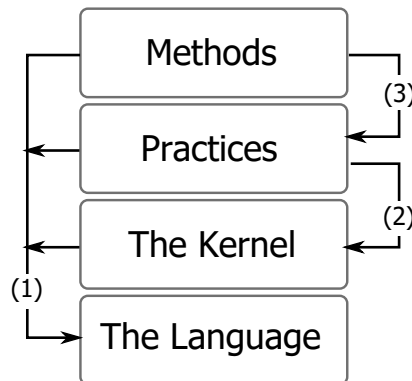


Figure 2.8: SEMAT Essence: Method Architecture[188]

The *Language* delivers syntactic infrastructure to define essential concepts in a *Kernel* (1). All elements and concepts are described using this domain specific language (1).

“The *Essence Kernel* captures the essential elements of software engineering, those that are integral to all software engineering methods.”[188]

Based on the concepts of the *Kernel*, *Practices* get defined (2). “A *practice* is a repeatable approach to doing something with a specific objective in mind. A practice provides a systematic and verifiable way of addressing a particular aspect of the work at hand. A Practice can be part of many methods.”[188]

A chosen *Kernel* and a set of chosen *Practices*, which got identified by a team as best fitting into the given context, eventually get composed into a *Method* (3). “*Methods* are not just descriptions for

developers to read, they are dynamic, supporting their day-to-day activities. [...] A method is not just a description of what is expected to be done, but a description of what is actually done.”[188]

2.7.2 SEMAT Essence Kernel

“The Software Engineering Kernel is a stripped-down, light-weight set of definitions that captures the essence of effective, scalable software engineering in a practice independent way.”[188]

Osterweil[192] stated, “Software processes are software too.” This statements fits very well with the software design principle *Separation of Concerns*³⁹, which the authors of *Essence* applied to SE methods. They separated the common ground, a stable kernel, from varying practices. And they separated the essentials of every SE endeavor from the details, depending on the respective context and the chosen set of practices.

The objective of the kernel is not to represent each detail of any SE method one could imagine. It was designed to be as small and universal as possible to represent the inherent essence of each SE endeavor. With that common foundation, practices can be defined, exchanged and applied independently. They can be cherry-picked, mixed and matched by project teams, organizations, and communities based on their unique needs. By combining the kernel and a set of chosen best (fitting to the context) practices, teams form their own *methods*. This method composition is illustrated in figure 2.9.

“You have achieved perfection not when there is nothing left to add, but when there is nothing left to take away.”

—SEMAT authors citing Antoine de Saint-Exupéry

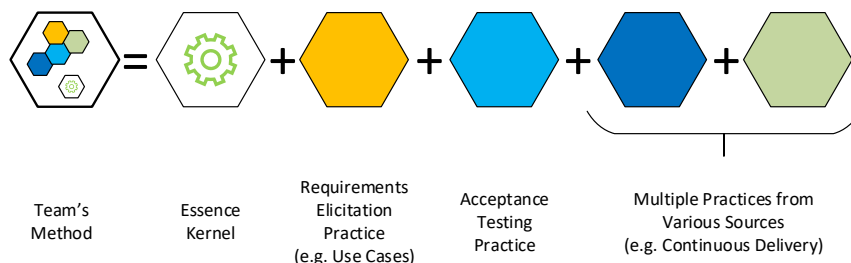


Figure 2.9: SEMAT Essence: Method Composition

To provide the essential elements of any SE endeavor, *Essence* uses the concepts of *Alphas*, *Activity Spaces*, and *Competencies*. To preserve its independence from any practice, it omits the inclusion of other elements, like *work products* and *activities*. These elements are re-

³⁹ “Separation of concerns is a principle that you can apply when designing software systems to create systems that are made of nonoverlapping modules, each handling a distinct concern. Separation of concerns helps software systems become extensible and maintainable. Here, we apply the principle of separation of concerns (SoC) to methods [...]”[118]

Area of Concern	Alphas	Activity Spaces	Competencies
Customer	Stakeholders, Opportunity	Explore Possibilities, Understand Stakeholder Needs, Ensure Stakeholder Satisfaction, Use the System	Stakeholder Representation
Solution	Requirements, Software System	Understand the Requirements, Shape the System, Implement the System, Test the System, Deploy the System, Operate the System	Analysis, Development, Testing
Endeavor	Team, Work, Way of Working	Prepare to do the Work, Coordinate Activity, Support the Team, Track Progress, Stop the Work	Leadership, Management

Table 2.17: SEMAT Essence Kernel: Elements By Area Of Concern

served for practices, which link their elements to kernel's respective elements. This is illustrated in figure 2.13 on page 103.

To facilitate orientation inside of the kernel, it is structured into three discrete *areas of concern*, each focusing on specific aspects of SE:

- *Customer*—focusing on the actual use, the motivation to build or change, and value provided by the software system to be produced.
- *Solution*—focusing on the specification and development of the software system.
- *Endeavor*—focusing on the team and how they approach their work.

Table 2.17 gives an overview of the elements contained in each of the areas of concern.

Alphas—The Things to Work With

Alphas are the things to work with in *each* SE endeavor.

Alpha is an abbreviation, standing for “*Aspiration Led Progress and Health Attribute*”[118] or “*Abstract Progress and Health Attribute*”[188]. Both terms stress that these elements represent attributes, or dimensions, of an SE endeavor, a team continuously needs to monitor for progress and health. The first version emphasizes goal orientation while latter emphasizes abstractness.

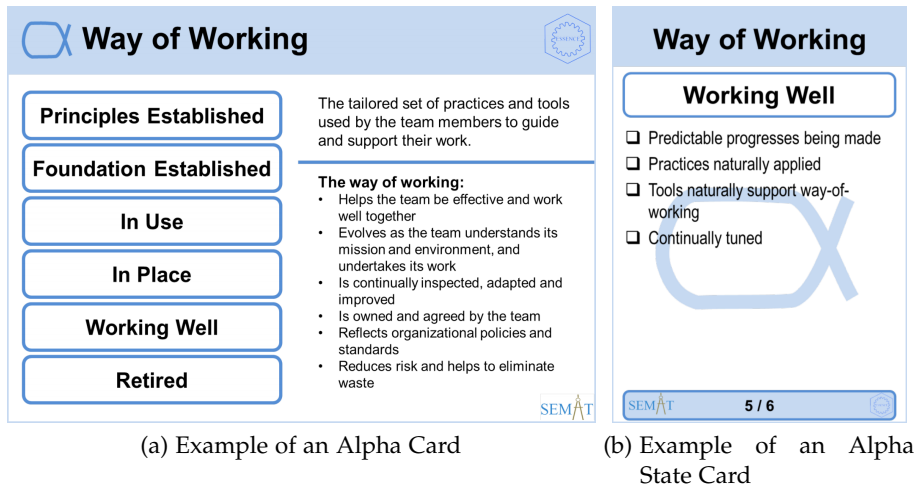


Figure 2.10: SEMAT Essence: Kernel Made Tangible by Provided Cards[280]

Depending on the individual perspectives taken so far, the *Opportunity* Alpha may appear more or less obvious to SE practitioners at first glance, since it represents a less “technical” element of daily SE activity. This holistic view on SE endeavors, including its surrounding and founding context, is a strength of the SEMAT Essence kernel—one that fits very well with *lean thinking* principles and concepts gaining popularity recently.

The Essence Kernel is made *tangible* by providing a *card metaphor*. SEMAT provides a set of printable cards containing Alphas’ definitions and Alpha States’ checklists. Such a set of cards is easy to carry around and supports agile team sessions. Figure 2.10 shows an example of both an Alpha card and an Alpha State card.

Alphas provide an abstraction over work products, found in other approaches, e.g. SPEM[186] or ISO24744[114]).

With this type of elements, it gets possible to provide a common kernel that is not bound to concrete practices, in which concrete work products—with specific practice-dependent designations and semantics—exist. Concrete *Work Products* are provided in Essence by *practices* defined on top of the *kernel* and each associated with a respective *Alpha* (cf. figure 2.13 on page 103). *Work Products* are defined with a set of *Levels of Detail*, similar to the *Alpha States*, and a corresponding lists of *Checkpoints*, supporting their assessment. By associating each *Work Product* with an *Alpha*, they are not just one of the artifacts to produce but contribute to an endeavor’s progress in a very clearly visible way.

Alpha	Definition	States
Stakeholders	<i>The people, groups, or organizations who affect or are affected by a software system.</i> “[...] provide the opportunity and are the source of the requirements and funding for the software system. The team members are also stakeholders. As much stakeholder involvement as possible throughout a software engineering endeavor is important to support the team and ensure that an acceptable software system is produced.”[188]	Required Represented Involved In Agreement Satisfied for Deployment Satisfied in Use
Opportunity	<i>The set of circumstances that makes it appropriate to develop or change a software system.</i> “[...] articulates the reason for [...] the software system. It represents the team’s shared understanding of the stakeholders’ needs, and helps shape the requirements for the new software system [...]”[188]	Identified Solution Needed Value Established Viable Addressed Benefit Accrued

Table 2.18: SEMAT Essence Kernel: Customer Area of Concern—Alphas[188]

The SEMAT Essence Kernel provides seven *Alphas*: *Stakeholders*, *Opportunity*, *Requirements*, *Software System*, *Team*, *Work*, and *Way of Working*. These *Alphas*, their definitions, and *Alpha States* are summarized in table 2.18 (*Customer* area of concern), table 2.19 (*Solution* area of concern), and table 2.20 (*Endeavor* area of concern).

Each *Alpha* is provided with a set of *Alpha States*. Each *Alpha State* owns a checklist supporting its assessment (cf. figure 2.11). An SE endeavor is progressed by progressing the states of all *Alphas* and retaining their health. The checklist provided to assess each of the *Alpha States* provides immediately usable expert knowledge enabling even less experienced teams to think holistically about the endeavor and recognize aspects that otherwise, based solely on team members’ experiences, might be forgotten. Teams are free to add checkpoints to the list as needed. This contributes to a continuously improving way of working (*kaizen*), promoted by *lean thinking* concepts.

Figure 2.12 illustrates the fact that *Alphas* do not exist in isolation. The figure depicts the *Alphas* of the Essence kernel and their associations. *Alphas* form a net of interrelated elements that have to be progressed in a balanced way. It is not possible to focus solely on one or a subset of *Alphas* if a team strikes for a healthy and progressing SE endeavor.

By utilizing the Essence kernel, organizations’ SE endeavors, which may be using different sets of practices in varying contexts, get comparable at the level of kernel’s *Alphas*. This facilitates organization’s

Alpha	Definition	States
Requirements	<i>What the software system must do to address the opportunity and satisfy the stakeholders.</i>	Conceived Bounded Coherent Acceptable Addressed Fulfilled
	<i>“It is important to discover what is needed from the software system, share this understanding among the stakeholders and the team members, and use it to drive the development and testing of the new system.”[188]</i>	
Software System	<i>A system made up of software, hardware, and data that provides its primary value by the execution of the software.</i>	Architecture Selected Demonstrable Usable Ready Operational Retired
	<i>“The primary product of any software engineering endeavor, a software system can be part of a larger software, hardware, or business solution.”[188]</i>	

Table 2.19: SEMAT Essence Kernel: Solution Area of Concern—Alphas[188]

Alpha	Definition	States
Team	<i>A group of people actively engaged in the development, maintenance, delivery, or support of a specific software system.</i>	Seeded Formed Collaborating Performing Adjourned
Work	<i>Activity involving mental or physical effort done in order to achieve a result.</i>	Initiated Prepared
	<i>“[...] everything that the team does to meet the goals of producing a software system matching the requirements, and addressing the opportunity [...]”[188]</i>	Started Under Control Conclude Closed
Way of Working	<i>The tailored set of practices and tools used by a team to guide and support their work.</i>	Principles Foundation Established In Use In Place Working well Retired
	<i>“The team evolves their way of working alongside their understanding of their mission and their working environment. As their work proceeds they continually reflect on their way of working and adapt it as necessary to their current context.”[188]</i>	

Table 2.20: SEMAT Essence Kernel: Endeavor Area of Concern—Alphas[188]

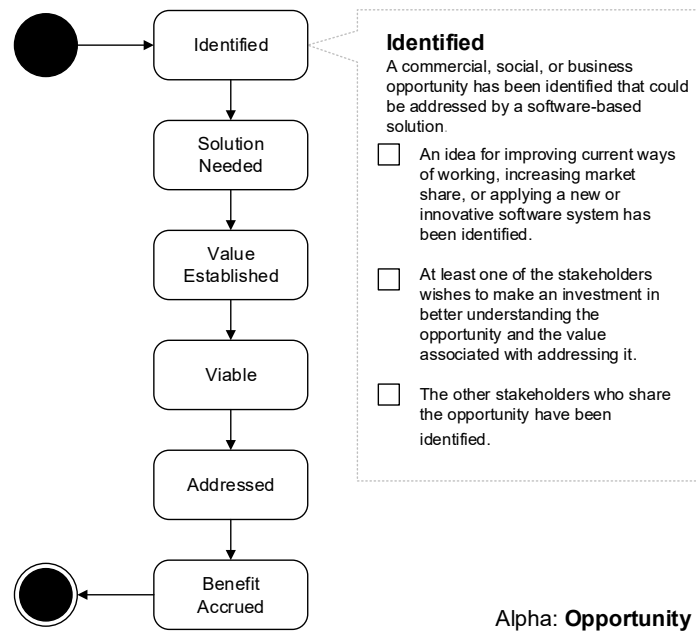


Figure 2.11: SEMAT Essence Kernel: *Opportunity* Alpha with States and Checkpoints for the *Identified* Alpha State

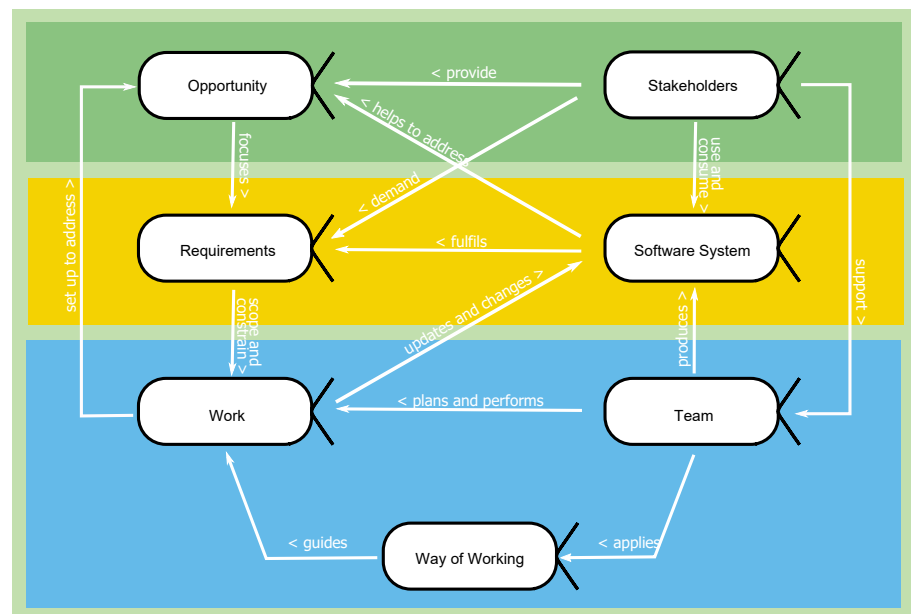


Figure 2.12: SEMAT Essence Kernel: Alphas and Their Associations [Screenshot of Essence Kernel Puzzler (cf. section 4.5) adapted from 119, 188]

tainments, knowledge, and skills necessary to do a certain kind of work.” This definition is quite similar to the one given in section 2.3 on page 38. The Essence kernel provides a definition for each of the six competencies (cf. table 2.17 on page 98) defined and adds a set of levels. These levels are: 1—*Assists*, 2—*Applies*, 3—*Masters*, 4—*Adapts*, and 5—*Innovates*.

Higher competency levels build on lower ones. “Individuals at levels 1 and 2 have an awareness or basic understanding of the knowledge, skills, and abilities associated with the competency. However, they do not possess the knowledge, skills, and abilities to perform the competency in difficult or complex situations and typically can only perform simple routine tasks without direction or other guidance. Individuals at level 3 and above have mastered this aspect of their profession and can be trusted to integrate into, and deliver the results required by, the team.”[188]

However, the Essence kernel defines no associations between Activity Spaces and Competencies. This is left to the provider of a particular practice.

2.7.2.1 *Special Characteristics of the Essence Kernel*

Three design goals make the Essence kernel special[118]:

1. It is designed to be *practical*: “The rationale for the kernel is to make the lives of software professionals easier by making methods more practical and intuitive.”[118] It encourages teams to take control of their method themselves and evolve and improve it as their endeavor progresses. Essence takes the stance that “Process engineers should serve the needs of software professionals, not the other way around.”[118] This is a big paradigm shift compared to other attempts of unified process definition, e.g. SPEM[186] or ISO24744[114].
2. It is designed to be *actionable*: Instead of just describing an SE method, the kernel provides support to actually use the method dynamically while working on an SE endeavor. *Alphas* with their *Alpha States* and corresponding checklists enable teams to monitor progress and health of the whole endeavor continuously. *Alphas* dynamically change their current state as the endeavor progresses. *Activity Spaces* addressing *Alpha States* provide support at defining next activities to progress the endeavor.
3. It is designed to be *extensible*: The kernel provides the common ground that practices can build on. Practices may provide additional (sub-)alphas, work products, and concrete activities, etc. to extend the kernel. Practices provide *How* to do things in an endeavor. Depending on team’s or organization’s capability and the background of its developers, the chosen practices may

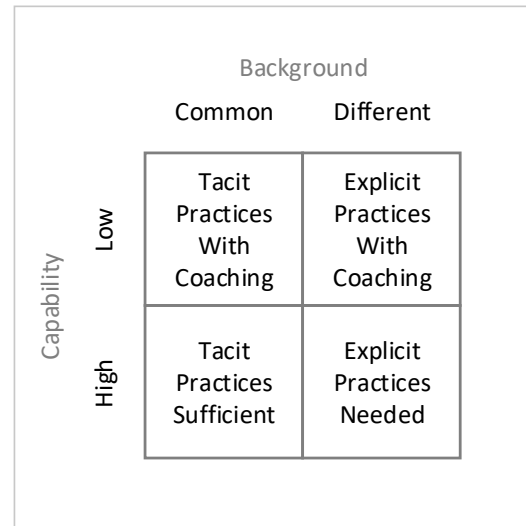


Figure 2.14: SEMAT Essence: Need for Practices to Extend the Kernel[118]

deliver more or less detail, just as needed (cf. figure 2.14). As practices are concerned just with single aspects of software development, they can be combined to the specific needs of teams and organizations. This is illustrated in figure 2.13 on page 103. The kernel provides the common ground that single practices, focusing various single aspects, can relate to, forming a consistent whole and resulting in a tailored method. A practice is not limited to the seven Alphas provided by the kernel. It can bring in its own Alphas or define sub-Alphas that define existing ones at a more fine-grained, practice-specific, level. Figure 2.15 illustrates that. The specification provides examples and proposes extensions to the kernel, which might be of use for a number of teams. Park et al.[198] provide a very detailed description of how to describe Scrum[141] in terms of and with the power of Essence.

Utilizing Essence to Drive an SE endeavor

Essence and its kernel support a highly structured, quantifiable, and goal-oriented approach to progress an SE endeavor. A typical proceeding follows a Plan-Do-Check-Adapt cycle⁴⁰, illustrated in figure 2.16:

1. *Plan*: A team starts with assessing the current state of the endeavor. *Alphas* and their *Alpha States* ensure that all relevant

⁴⁰ The *Plan-Do-Check-Adapt cycle* represents an adapted *Demen* or *Shewart cycle*[168], well known in the field of modern quality control and continuous improvement and as such part of *lean thinking* principles.

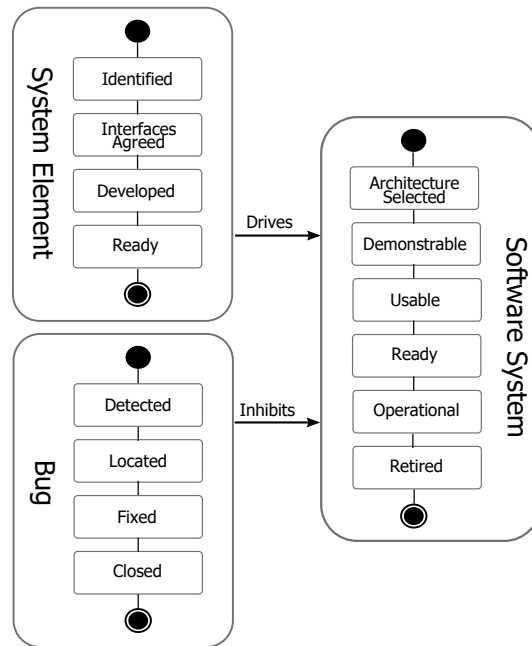


Figure 2.15: SEMAT Essence: Two of Development Extension's sub-Alphas: *Bug* and *System Element* inhibiting and driving, respectively, the *Software System Alpha*

dimensions are considered. *Checkpoints* support the assessment of each of the *Alpha States*. *Walktroughs* or *Assessment Poker*⁴¹ sessions are suitable to assess an endeavor in a team session.[118] The *Activity Spaces* addressing next targeted *Alpha States* support by identifying next activities. *Checkpoints* of next planned *Alpha States* help to identify necessary tasks.

2. *Do*: The team works on accomplishing the identified tasks. Emerging obstacles are removed as they occur.
3. *Check*: By continuously tracking progress toward defined objectives and tasks, the team ensures to keep focused and following its chosen way of working.
4. *Adapt*: Team's way of working is continuously reviewed and assessed in the same way as all other essential dimensions (*Alphas*) of the endeavor. As a team recognizes impediments caused by their way of working, better (suited) ways of getting things done get identified and used. Plans are adapted accordingly.

Essence Language Meta-Levels and Method Enactment

The Essence language is utilizing UML language infrastructure[191] and consists of a set of meta-levels:

⁴¹ Referring to *Planning Poker* sessions supporting teams using agile methods, e.g. Scrum[141].

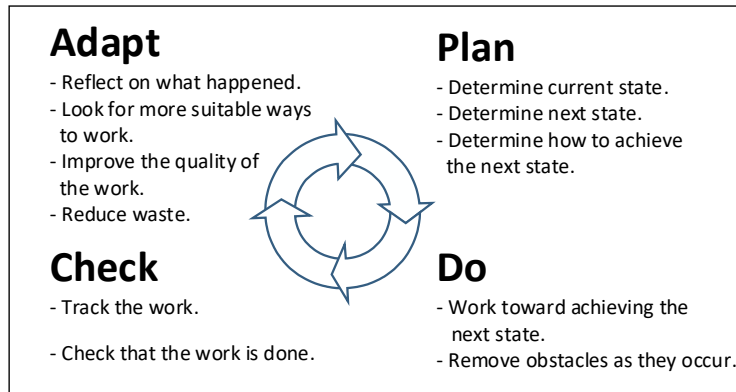


Figure 2.16: SEMAT Essence Kernel: Plan-Do-Check-Adapt Cycle[118]

- *Level 3 (M3)*—holding the meta-language to define the basic element types. (UML MOF⁴²)
- *Level 2 (M2)*—holding Essence’s language constructs, e.g. *Alpha* and *Activity*.
- *Level 1 (M1)*—holding the specification elements representing the kernel and practices, instances of M2, e.g. *Requirements Alpha*, *Stakeholders Alpha*, *Explore Possibilities ActivitySpace*, etc.
- *Level 0 (Mo)*—holding the instances (occurrences) at runtime of an endeavor, e.g. the unique *Software System Alpha* of *Team B’s endeavor in an SE course in summer semester 2016*.

The *Enactment* of an Essence Method is defined in the *Dynamic Semantics* of the specification. Essence does not provide only a static method description. It defines how to enact an Essence method (at level M1) to provide all necessary elements and structures (at level Mo) to run an endeavor and to track the progress and express the overall state of a running endeavor. The semantics of a guiding function supporting a team to plan next tasks to do after defining next target state of the endeavor are provided too.

Section 4.6 on page 217 describes a tool that provides Essence method enacting and support at progressing an endeavor.

Essence AND Traditional Methods and Improvement Frameworks

McMahon[199] states “Essence is not in competition with any method or improvement approach. It can help your team implement whatever approach you choose by helping them ask the right questions leading to better choices based on where their team is now and where

⁴² The UML *Meta Object Facility*, part of the UML infrastructure, used to define the UML—and Essence.[191]

they need to go next. [...] it can help your team reduce the risk of failing to achieve a successful outcome and it can do it independent of whatever way of working you have chosen.”

He adds “the comparisons—I suggest—should not be Scrum versus Essence, or CMMI versus Essence, or RUP versus Essence. But rather they should be Scrum versus (Scrum + Essence), or CMMI versus (CMMI + Essence), or RUP versus (RUP + Essence).”

While the Essence language can be used to implement existing SE methods, it “is not in competition with any existing practice or method. It is agnostic to your chosen approach and can be implemented by your team today without changing what you are currently doing. It is more about helping you do what you are already doing so you can keep doing it even better in the future.”[199]

By adding a common ground, or a “reference model”, that all teams can use, regardless of their chosen (agile or not) method, Essence adds to the ability to constantly measure health and progress of endeavors and to improve teams’ way of working.[120, 124]

2.7.3 *Essence In Education*

The Essence kernel was already introduced in various courses all over the world[118, 184, 206, 207] resulting in encouraging findings. The usage of Essence in education and its contribution to learning are discussed in section 4.2 on page 189.

2.7.4 *Conclusions*

With Essence the SEMAT initiative provides an approach enabling a shift towards flexible and composable SE practices instead of monolithic software processes prevalent in past and presence.

With the Essence kernel, this standard provides a compact, practical, actionable, and extensible set of elements inherent in all SE endeavors. As such this kernel serves as a thinking framework as well as a foundation to define practices and compose them to tailored methods providing teams with an efficient way of working.

Since the kernel is actionable, it enables a team to continually and holistically monitor progress and health of all essential dimensions in an efficient way. With utilizing a PDCA cycle to steer the SE endeavor the kernel provides the foundation for continuous improvement throughout an SE endeavor.

With this holistic and pragmatic approach and the associated high transferability of concepts and knowledge it provides important characteristics that indicate its suitability in SE education. This gets confirmed by results of first deployments in academic education and is why SEMAT Essence was chosen as foundation of the *Integrated Ap-*

proach introduced in chapter 4 on page 179.

Binding this approach to the SEMAT Essence specification might not be without risks. As already discussed, Essence provides strong arguments for its adoption. Nonetheless, new standards are questioning old ones. As people and organizations, in the business, as well as in the academic world, may be invested in utilizing specific processes, methods, etc., there may exist a considerable resistance to a prompt adoption.

2.8 SOFTWARE PROCESS SIMULATION MODELING (SPSM)

The preceding section explored the field of software processes, methods, and practices. To predict outcomes of using or changing existent processes as well as generally gaining more insight of the dynamics of software processes and projects the field of *Software Process Simulation* (SPS) and *Software Process Simulation Modeling* (SPSM) have been developed and applied.

Kellner et al. describe “A *software process simulation model* focuses on some particular software development/maintenance/evolution process. It can represent such a process as currently implemented (as-is), or as planned for future implementation (to-be). Since all models are abstractions, a model represents only some of the many aspects of a software process that potentially could be modeled—namely the ones believed by the model developer to be especially relevant to the issues and questions the model is used to address.”[139]

Zhang et al. state “Process simulation has become a powerful technology in support of software project management and process improvement over the past decades.”[311] A considerable body of literature was published over the past decades in this field.[11] Since the suggestion to use simulation modeling to gain a better understanding of the software development process, attributed to the work of McCall et al.[164] in 1979, and pioneering work in the 1980s[2, 3], SPS “has been applied in many software development/maintenance projects with varying process scales and organization settings over the past decades. It leverages planning, managing, controlling, improving software processes, and provides software practitioners (professionals and managers) powerful tools and recognizable benefits.”[311] Since then hundreds of studies on process simulation for SE had been conducted, a number of secondary studies scoping available research were published, and several conferences and workshops encourage contributions in this field.[311]

2.8.1 *Motivation/Purpose/Objectives/Scope*

Kellner et al. state that there “is a wide variety of reasons for undertaking simulations of software process models.”[139] They clustered purposes of utilizing simulations of software processes into six categories. Zhang et al.[312] suggest a more fine-grained classification into ten purpose categories, grouped into the three levels *cognitive*, *tactical*, *strategic*:

- *cognitive* level
 1. understanding,
 2. communication,
 3. process investigation,
 4. training and learning,
- *tactical* and *strategic* levels (differing in scope and impact):
 1. prediction and planning,
 2. control and operational management,
 3. risk management,
 4. process improvement,
 5. technology adoption,
 6. trade-off analysis and optimizing.

As with every modeling attempt, the purpose, and the questions, which one likes to get answered, are determining model’s scope.

Madachy states “The simulation process in an organization involves designing a system model and carrying out experiments with it. The purpose of these ‘what if’ experiments is to determine how the real or proposed system performs and to predict the effect of changes to the system as time progresses. The modeling results support decision making to improve the system under study, and normally there are unintended side effects of decisions to consider. The improvement cycle continues as organizational processes are continually refined.”[159]

The scope of such simulation models “range from *single phase*, *multi-phase*, *project*, to *product evolution*.”[311] Depending on research interest the output variables of simulation models may vary. Zhang et al.[311] identified *time (duration)*, *effort (cost)*, *quality*, *(software) size*, and *resource* as the most interesting indicators.

Kellner et al. state “A simulation can be deterministic, stochastic, or mixed. In the deterministic case, input parameters are specified as single values (e.g., coding for this unit will require 5 work-days of effort, or 4 hours per hundred lines of code; there will be two rework cycles on code and unit test; etc.). Stochastic modeling recognizes the inherent uncertainty in many parameters and relationships.

Rather than using (deterministic) point estimates, stochastic variables are random numbers drawn from a specified probability distribution. Mixed modeling employs both deterministic and stochastic parameters.”[139] Purely deterministic models deliver the same result at each run, so only one run is needed for a set of parameters to obtain those results. Results of stochastic and mixed models are likely to vary from run to run. These results may be analyzed statistically across a batch of simulation runs (*Monte Carlo simulation*).[139]

Building and using software process simulation models is not a cheap endeavor. As Zhang et al. state “The development effort required for the most complex software process models may reach up to a few person years.”[313] Ali et al. criticize that most studies do not report the effort associated with their conducting, what interferes efficiency analysis of the approaches taken.[11] They cite from rare studies that reported efforts of more than one person-year of effort for such SPSM studies. They add “Given the nature of software development where change is so frequent [...] apart from the cost in terms of required tool support, training and effort for development, use and maintenance of simulation models there is the cost of the necessary measurement program that can feed the model with accurate data that should also be acknowledged for an effective process simulation.”[11]

Given such high investments, Zhang et al. state “On the basis of recent evidence in the community [...] SPS remains a much-needed field of endeavor but it needs additional maturing. Although it can be very valuable in addressing many problems, it can be more than what is needed for some problems and can be inadequate for others.”[313] They propose to set expectations and prediction accuracy as well as precision in parameter values into relation with the context of the modeling purpose and desired results. They state “Using a simplified model with estimated parameter values can often provide useful results (even if not exact or proven) and process insights with a limited investment of time and resources and is far better than not simulating. Trustworthy, quantitative predictions while obtainable for certain applications require a higher level of investment and remain challenging to achieve in some aspects of software engineering.”[313]

Challenges Remaining

“However, the impact of SPS research is very difficult to quantify. Anecdotal evidence exists for the successful applications of process simulation in software companies.”[311] Basically studies are proclaiming benefits of SPSM in all the purpose areas aforementioned.

Ali et al. state “Such range of claimed potential benefits and reports of industrial application and impact give an impression that simulation is a panacea for problems in Software Engineering (SE). However, some authors have recently questioned the validity of these

claims.”[11] In their systematic literature review[11] they “aim to aggregate and evaluate, through a systematic literature review, the empirical evidence on the usefulness of SPSM in real-world settings (industry and open source software development)” and claim “A large majority of the primary studies scored poorly with respect to the rigor and relevance criteria.”

The validation and verification (V&V) of simulation models is interfered by a lack of empirical data. Dickmann et al. state “Methodologically, the simplest case is to prove congruence of the simulation results with real world input-output-data. However, this type of validation is almost never achieved because of ‘lack of data’ from software development projects. Moreover, this ‘lack of data’ is not due to the shortcomings of the respective research effort (which might be overcome by a better approach), but poses a principal methodological problem: software producing organizations are time-varying and can not be experimentally tested for different scenarios.”[71]

Zhan et al. mention the needed sets of skills that are rarely combined: “Successful industrial adoption of SPS requires four knowledge areas and skill sets: modeling and simulation, analytics and statistics, managing engagements, and software development and maintenance (i.e., domain knowledge and experience). This combination of knowledge and skills is difficult to find in industry. Many people are developed in one or two of these areas, but relatively few have developed three or all four of them.”[313]

Ali et al.[11] cite Pfahl, that “there is no evidence of wide spread adoption and impact of SPSM research on industry. He also challenges if SPSM has a realistic potential to have an impact on industrial practice.” The authors conclude a “need to identify the problems where the use of simulation can be justified given the high cost of undertaking it and show its utility.”[11] and recommend “In future work, based on our findings, it is important to evaluate simulation against the purposes (e.g. education and training, prediction), which has not been done so far. Future studies should not focus on evaluating the ability of simulation to reproduce reference behaviour, the fact that it is capable to do this is well established in the literature.”

2.8.2 *Prevalent Simulation Formalisms*

To investigate different aspects of software processes various modeling paradigms have been used. Kellner et al.[139] mention “Despite best intentions, the implementation approach often influences what is being modeled. Therefore, the most appropriate approach to use will be the one best suited to the particular case at hand, i.e., the purpose, questions, scope, result variables desired, etc.”

Modeling approaches available have different strengths. “In some instances, die-hard proponents of a given simulation modeling ap-

proach have seemed to argue that theirs is the best approach to use and is entirely appropriate for every situation. It is probably true that a model developer who is very skillful with a particular approach and tool can succeed in modeling almost any process situation with that approach and tool – no matter how awkward and unnatural the representation may ultimately be. However, we are convinced that no single modeling approach or tool is the most natural and convenient one to use in all software process situations.”[139]

Software processes have been simulated by a number of simulation formalisms/paradigms, including *discrete-event simulation* (or *state-based simulation*), *System Dynamics* (or *continuous simulation*), *hybrid simulation* approaches, and less often *Petri nets*, *agent-based simulation*, and further approaches. *System Dynamics* (SD) and *Discrete-Event Simulation* (DES) have been the two most commonly used simulation paradigms.[7, 9, 11, 139, 154, 311]

2.8.2.1 *System Dynamics* (SD)

So far System Dynamics (SD) has been the most utilized simulation paradigm in SPSM. It was this paradigm that was used by Abdel-Hamid to build first simulation models of the dynamics of software projects[2, 3]. Lots of other models and approaches to utilize SD in SE education[49, 90, 210–212] followed over the years.

“Models using the system dynamics paradigm represent the project environment as a set of differential equations. Integrating these equations over time describes the behavior of project variables such as staff levels, motivation, and the number of detected errors.”[163]

Osterweil states those models are “focused on phenomenological observations of external behaviors of processes.”[193] As Kellner et al. describe “Continuous-time simulations (e.g., system dynamics) tend to be convenient for strategic analyses, initial approximations, long term trends, high-level (global) perspectives, etc. – essentially analyses above the detailed process level. [...] System dynamics [...] models the levels and flows of entities involved in the process, although those entities are not individually traced through the process. Changes happen in a continuous fashion.”[139] Time is advanced in relatively small constant steps. A modeler controls accuracy of calculations by setting the time steps small enough.[163]

Martin and Raffo state “While these models are very good at demonstrating the effects of feedback loops which may exist in the project environment, they are inherently limited in their ability to represent discrete process steps.”[163] Elements or entities in the process, such as project members or process artifacts, are modeled in aggregated form, omitting single steps in the process.

	System Dynamics	Discrete Event System
Focus	macro-level process dynamics ^a	micro-level process dynamics ^b
Time Orientation	continuous ^c	discrete ^d
Advantages	accurately captures the effects of feedback, clear representation of the relationships between dynamic variables	CPU efficient since time advances [solely] at events, attributes allow entities to vary, queues and interdependence capture resource constraints
Disadvantages	sequential activities are more difficult to represent, no ability to represent entities or attributes	continuously changing variables not modeled accurately, no predefined mechanisms for states and their transitions ^e

Table 2.21: Comparison of Modeling Techniques: *System Dynamics* vs. *Discrete Event System* [139, 311]

^a “focused on phenomenological observations of external behaviors of processes”[193]

^b “focused on the study of the internal details and workings of processes”[193]

^c represented by a set of differential equations

^d Simulation time advances solely at events, potentially changing state.

^e The DEVS formalism (cf. section 3.4) provides *internal* and *external transition functions* to define timing (*the what and when*) of its models but leaves mechanics (*the how*) to transition between (potentially complex) states to the modeler.

2.8.2.2 Discrete Event Simulation (DES)

Kellner et al. describe Discrete Event Simulation (DES): “Discrete event and state-based simulations tend to be convenient for detailed process analyses and perspectives, resource utilization, queuing, relatively shorter-term analyses, etc. [...] Discrete event models contain distinct (identifiable and potentially differing) entities that move through the process and can have attached attributes. Changes happen in discrete steps. This supports sophisticated, detailed analyses of the process and project performance.”[139] Such entities may include specific process tasks and unique process artifacts with descriptive attributes.[163]

The *Discrete Event System Specification (DEVS)* is a specific DES formalism with a sound theoretical background. It was introduced in the early 1970s[308]. Since then it was utilized in many different domains and extended over the following decades. Discrete Event System Specification (DEVS) gets described in section 3.4 on page 131.

Table 2.21 summarizes characteristics of SD and DES.

Hybrid Simulation Approaches

To make use of more than one simulation paradigm, they got combined in hybrid simulation approaches. Such approaches employ

more than one simulation technique, e.g. by combining discrete and continuous simulation. Zhang et al.[311] state “By combining discrete and continuous simulations, for example, hybrid process simulation is capable of addressing both the micro-level and macro-level process dynamics, and breaks through the limitations of applying any single simulation method. However, integrating these two approaches faces the issues of compatibility, interoperability and synchronization when executing simulation. Most of the hybrid process simulations are based on the combination of SD and DES.” A number of hybrid simulation models of software processes were developed by the SE community [54, 75, 158, 163, 220].

2.8.3 Conclusions

The two most common simulation formalisms for SPSM are SD and DES, both having their strengths and weaknesses. A simulation formalism should be chosen depending on its strengths related to the purpose the simulation should provide.

SD has its strength at capturing feedback loops and delays, both challenging and often not easy to grasp to the human mind. It is focused on the macro-level process dynamics. To represent sequential activities is not a strength of this approach.

The primary purpose of the simulation is to drive a digital learning game providing the opportunity to control a virtual team in an SE endeavor, basically to follow a *PDCA cycle* (cf. section 2.7.2.1 on page 105), and to provide a holistic view on the endeavor. This requires a focus on micro-level process dynamics and the representation of sequential activities.

DES provides those characteristics. DES can be implemented in several ways. For the introduced approach a formalism with strong theoretical foundation, one that is capable of providing hybrid simulation once it should be required, is preferred over a built-from-scratch approach. That’s why DEVS got chosen to provide its strengths to drive a new simulation approach supporting DGBL in SE (methods) education. This simulation approach gets introduced in the next chapter 3 on page 117.

A NEW SIMULATION APPROACH SUPPORTING DGBL IN SE METHODS EDUCATION

This chapter introduces a new simulation model designed to support DGBL in SE methods education. Before the model itself gets described, the experimental frame and guiding principles for a new modeling approach get defined. The foundational building blocks, the DEVS simulation formalism and Finite State Machines, get introduced. While the SE community might be familiar with the latter, the first one is rather special and likely less known for all those, less exposed to simulations and their formalisms. The simulation approach gets described in detail, and a streamlined modeling workflow gets introduced. The chapter closes with a conclusion and discussion of conceivable enhancements to the approach which might be topics of future work.

*“Everything should
be made as simple as
possible, but not
simpler.”*
—Albert Einstein

3.1 EXPERIMENTAL FRAME

This simulation approach aims at enabling interactive exploration of the SE methods¹ based on SEMAT Essence, underlying concepts, elements and their dependencies to provide the foundation of a game environment, where students of SE are enabled to control a virtual SE endeavor preparing them for their own real SE endeavors.

3.2 GUIDING PRINCIPLES FOR A NEW MODELING APPROACH

Synergies

It was already stated (section 2.5 on page 68) that existing simulation and game approaches in software process education require remarkable initial training effort to get familiar and productive using them—unfortunately those initial investments are not of direct use outside of the particular simulation and game environment.

It is often mentioned that the modeling process itself is an extremely instructive activity for any modeler because it makes the modeler thinking deeply about the problem domain. This research work aims at a straightforward modeling process that is highly transparent and enables direct knowledge transfer from the modeling process to practical application.

¹ for now focused on the SEMAT Essence kernel, but extensible to provide Essence methods in the future

Thinking about the simulation model should be to the greatest extent thinking about Essence itself. Adding as few concepts as possible to enable interactive exploration in a game context, the effort invested by users of this simulation approach should ideally be of direct use outside of the simulation and game environment—making the modeler a better Essence user or even designer. Limitations or ambiguities emerging modeling the simulation should be attributable to the chosen Essence method, practice or kernel itself. That way finding solutions to such challenges should enable the modeler to be aware of them in real SE endeavors, to apply similar solution strategies and to refine the underlying Essence kernel, practice or method itself.

Simplicity and Transparency

Following the philosophy of Essence, the simulation approach should be able to enable a holistic view to SE by containing only the essential elements. On the other hand, it should be extensible to contain more fine-grained and detailed elements when needed. Results of simulation runs should be easy to explain. Reducing embedded concepts to the essentials while keeping a holistic view reduces interfering effects obfuscating learning outcomes.

Having in mind that this simulation approach is primarily destined for integration into a game environment, players of such game should not be faced with the underlying simulation formalism. They should be focused on applying the chosen method/kernel and communicate with their virtual team without any further abstractions.

Integrated SE Method Description

The embedded method/kernel documentation serves as learning material. It is closely connected to additional tools utilized in the gameplay (cf. section 4.6 on page 217) and hence an integral part of the gameplay.

Efficient Model Construction and Customization/Flexibility

Building a simulation model following this proposed approach should be a transparent process. Utilizing Essence method definitions, defined in a standard tool, the effort to enhance this model with parameters for the intended purpose should be straightforward and feasible for everyone familiar with essence concepts. If no deep customization of the underlying simulation model elements' internals is needed, the modeler should not need any deeper knowledge about simulation formalisms. Anyone in need of deeper customization of the underlying model elements should find a simulation model based on approved standards and enabled to be relatively easily extended.

Deterministic simulation model

Primarily intended for supporting a game environment that facilitates social interaction as collaboration and competition the simulation model has to be deterministic and always to deliver same results given the same inputs. The *Simulation Game* based on this simulation model is intended to be played more or less just once by students to prepare them for practical project work, hence the enhancement of the model with stochastic elements would not provide the variation that might be desirable when the simulation/game would be run multiple times by the same learners.

Interactive and Embeddable in a Game Context

As stated this approach is designed to produce interactive models which primarily serve as the foundation of a game environment. Most simulation libraries available are not built to provide interactivity out of the box but to run a simulation once and to analyze the collected measures.

Models resulting from this proposed approach are built to provide interactivity. A person running the simulation is enabled to assign activities to a virtual workforce (Essence Alpha *Team*). The running simulation model provides feedback upon the occurrence of defined events, e.g. the start or the accomplishment of a given task. Integrated simulation model development and execution environments comparable to IDEs in software development are not designed to be embedded in such a game context.

The simulation model and its execution environment have to be able to communicate with a game environment, taking input from and delivering output to this game environment. Communicating with a web-based game environment requires corresponding communication mechanisms. The execution environment has to be able to run multiple simulations simultaneously to enable a course of SE students to play at the same time fostering social interaction.

Extensible and Testable

The approach was designed to be initially capable of the SEMAT Essence kernel but to provide the opportunity to be extended, e.g. to represent dynamic Essence methods composed of Essence practices.

Building software to embody a simulation model to depict SE processes, the elements of the model and the model itself have to be testable by automatic test or specification suites.

Evaluation

The evaluation of the simulation model is provided by its reasonable utilization by the game environment it is designed for. Different from industrial use cases (cf. section 2.8 on page 109), the simulation model is not designed to predict future outcomes of software projects in some way. Hence a fit of models results with historical project data is not rewarding.

3.3 BUILDING INTERACTIVE INSTRUCTIONAL MODELS BASED ON SEMAT ESSENCE

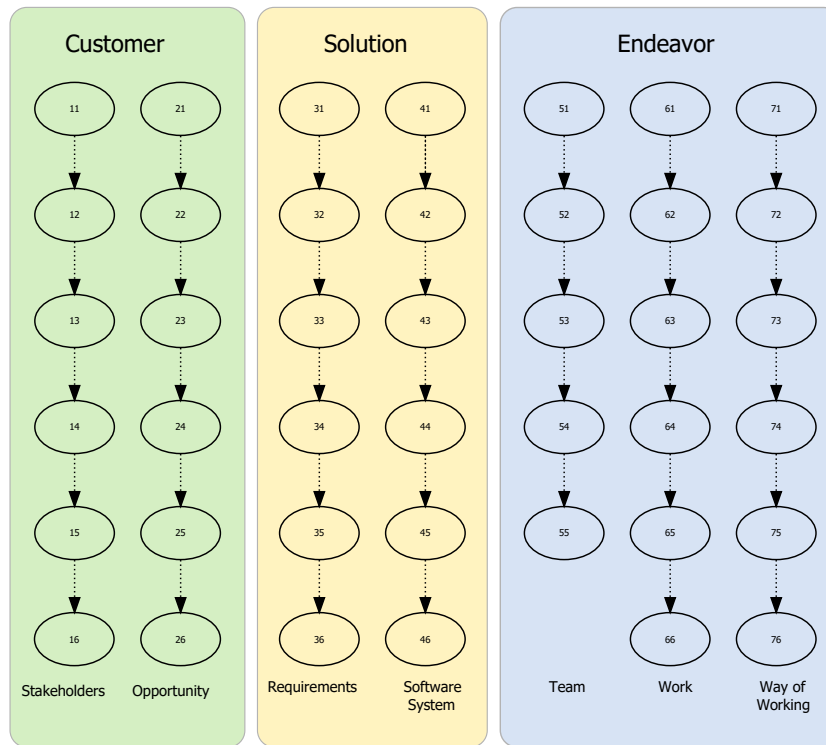
The simulation approach presented here uses the elements of the SEMAT Essence language especially the elements of the Essence kernel described in detail in chapter 2.7. The basic idea is to take these elements like Alphas, Alpha States, Checkpoints and ActivitySpaces from a defined Essence kernel and to build an instructional simulation model out of them enabling their interactive exploration while taking account of the inherent conditions and (inter-)dependencies of method's elements.

Making Alpha State (Inter-)Dependencies Explicit

This section describes dependencies and interdependencies of Essence kernel's Alpha States. To comprehend the elements of the simulation model and their behavior it is important to be aware of the different kinds of (inter-)dependencies inherent in the Essence kernel.

Simple Alpha Progressing

As described in section 2.7 an Alpha progresses through its defined set of Alpha States. Figure 3.1 on page 121 shows the progressing of the seven Alphas of the Essence kernel. In this diagram, the Alphas are grouped by their containing Area of Concern. For the sake of brevity the Alpha States are coded as numbers following a coding scheme described in appendix A on page 325. Each column represents the states of one Alpha. Once all Checkpoints of the first Alpha State are fulfilled the state is reached, and the Alpha may progress to its next Alpha State. This process repeats until the target state or the final state is reached. Each Alpha State, except the respective first state of an Alpha, is dependent on its predecessor state. This is represented by the dotted edges. To reach an Alpha State, all its checkpoints have to be fulfilled. This gets accomplished by performing an Activity which targets at reaching that state which is shown in Figure 3.3 on page 123.



cf. Appendix A on page 325 for the coding scheme of Essence kernel elements used in graphs

Figure 3.1: Alpha Progressing of the Three Areas of Concern

Alpha State Reassessment – lost Checkpoints and Alpha States

Actually, this is not yet the complete picture because as Alpha States may get reached, they can get lost too. This is exemplarily pictured for the Alpha Team in figure Figure 3.2 on page 122. Once the state *Performing* is reached, the team is enabled to work effectively and efficiently. But if a reassessment revealed that at least one of the checkpoints of this or any of the prior states would not be fulfilled anymore, that state got lost and the current state of the Alpha Team is defined to be the most advanced state where all of the checkpoints are fulfilled. In that case, the team would be missing some condition to tap its full potential. This case is depicted by the dotted arrows leading back to one of the prior states in figure Figure 3.2 on page 122.

In an extreme case, an Alpha could lose all its states given that just one checkpoint of its first Alpha State got lost. To remove obstacles in the way that led to the lost checkpoint(s) should be a highly prioritized task for a team in an SE endeavor. In the example given insufficient attention paid to one or more of the ongoing activities leading to the reached AlphaState *progressing* may cause such lost Alpha States. Stopping to *Coordinate Activity*, *Support the Team* or *Track Progress* would inevitably lead to losing this Alpha State and a lower overall performance of the team.

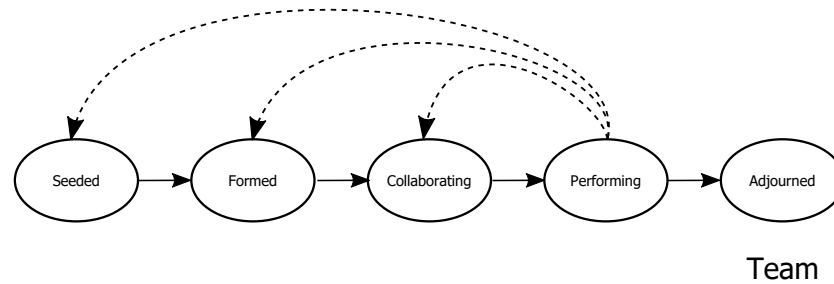
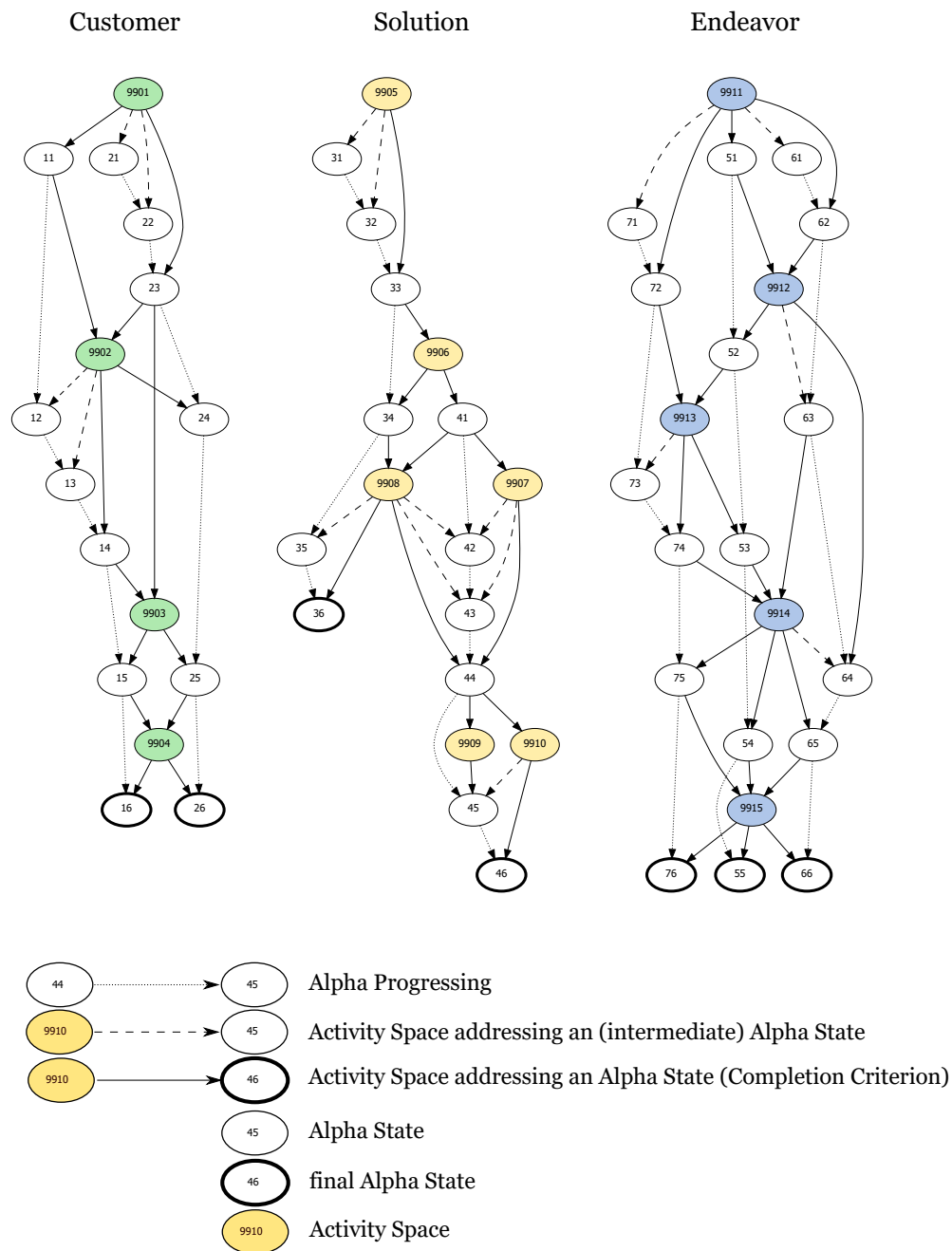


Figure 3.2: Progressing and Potential Retrogressing of the TEAM Alpha

Diagram Figure 3.3 on page 123 supplements the progressing of Essence Kernel's Alphas by depicting Activity Spaces and their relationships to the states of kernel's Alphas. For brevity reasons, Alpha States and the Activity Spaces are again coded as numbers following a coding scheme described in the appendix A on page 325. Dotted edges in the diagram show the regular progressing of Alphas defined in the Essence kernel as seen in Figure 3.1 on page 121. The colored nodes in the graph represent Activity Spaces which address Alpha States represented by the non-colored nodes. Each dashed and solid directed edge in the graph represents a dependency between an Alpha State and an Activity Space. Solid edges incoming to an Activity Space represent entry criteria as defined in the Essence specification version 1.1 [188]. These entry criteria were added in version 1.1 of the standard. Solid edges outgoing from an Activity Space represent the completion criteria of that Activity Space as defined by the Essence specification [188]. Version 1.1. of the standard defines only the most advanced state of an Alpha addressed by the Activity Space explicitly and omits the intermediary states addressed by the Activity Space leading to the state defined as completion criterion. Version 1.0 [187] of the standard was more explicit at stating that relation. Intermediary states addressed by the Activity Space are represented by dashed outgoing edges in the graph shown in Figure 3.3 on page 123.

The relation between an Alpha State and an Activity Space is not necessarily a one to one relation. Figure 3.4a on page 124 shows a zoomed in portion of the graph to make apparent that one Activity Space may address multiple Alpha States including states of different Alphas and intermediary states leading to the state defined as completion criterion. In this example the Activity Space *Explore Possibilities* targets at reaching Alpha States of two Alphas, firstly state *Identified* of the Alpha *Stakeholders* and secondly state *Value Established* of the Alpha *Opportunity* (via the preceding states *Identified* and *Solution Needed*). On the other hand Figure 3.4b on page 124 visualizes that one Alpha State may be addressed by multiple Activity Spaces. The example represented by the zoomed in portion of the graph shows that the states *Demonstrable*, *Usable* and *Ready* of the Alpha *Software*



cf. Appendix A on page 325 for the coding scheme of Essence kernel elements used in graphs

Figure 3.3: Alpha Progressing of the three Areas of Concern

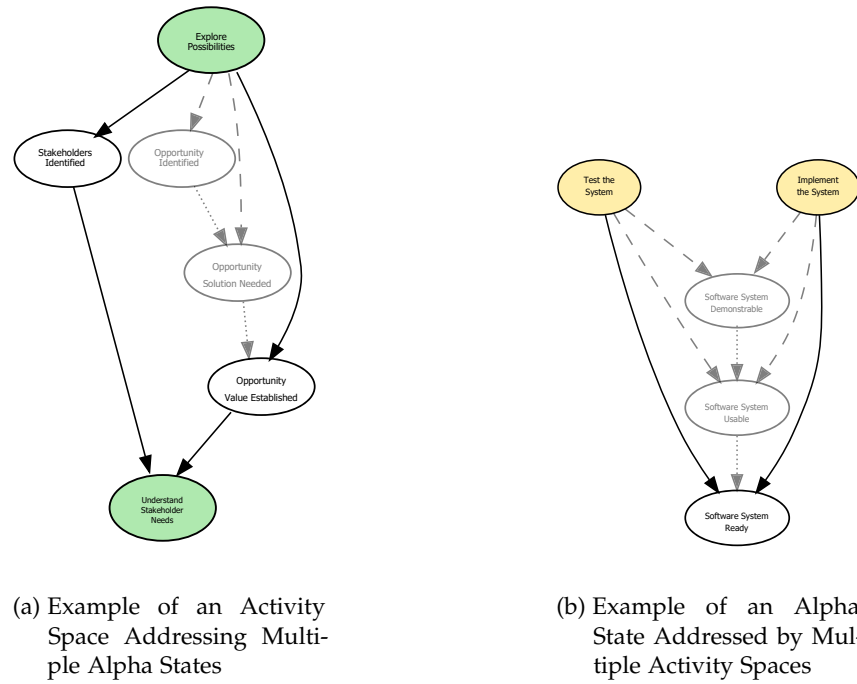


Figure 3.4: Examples of n:m Associations Between Alpha States and Activity Spaces

System require performing the two Activity Spaces *Implement the System* and *Test the System*.

Alpha State Interdependencies

In section 2.7 on page 95 was elaborated that the Alphas of the Essence kernel form a net of interrelated elements that have to be progressed in a balanced way. None of the Alphas exists in isolation and could be progressed to a final state solely on its own. Alpha Associations shown in figure 2.12 on page 102 clarify the dependencies between alphas. Alphas' progress is defined by their set of Alpha States. On a lower level, the dependencies between Alphas are apparent in interdependencies of Alpha States. To reach a state of one Alpha may require reaching a specific state of another Alpha beforehand. While those dependencies exist in the Essence kernel to provide a holistic view and the requirement of balanced progressing of Alphas they are not explicitly defined in the Essence specification. Dependencies are manifested in descriptions of Checkpoints defining the achievement of an Alpha State.

As extensively described in an educational case study[244], a team in an SE project would assess the current state of an SE endeavor utilizing Alphas, their states, and their Checkpoints. The team would define next goals regarding next target states and define next tasks to do—either derived from descriptions of the checkpoints to fulfill

next to achieve the target states or by utilizing the Activity Spaces addressing the defined target states.

Discussing single Alpha States and their interdependencies requires referencing of single Checkpoints describing them. To enable referencing single Checkpoints of Alpha States, they got numbered in B on page 329 section. Neither does this numbering exist in the Essence specification [188] nor does it define any sequence on them. It is used solely for identification and referencing purposes. The graph presented in figure 3.7 on page 130 shows the result of the analysis. Elements of the Essence kernel got coded as numbers following a coding scheme described in appendix A on page 325. Details of the semantical Checkpoint analysis get presented in appendix B on page 329.

To enable a simulation model based on the Essence kernel approach and to interactively explore an SE endeavor the inherent dependencies of Checkpoints' descriptions resulting in Alpha State interdependencies have to be made explicit and integrated into the model to provide a realistic perspective.

It is not the goal of this work to define the ONE correct path through an SE endeavor but to define a set of plausible ones that is possible to explain. There is not just one possible path. As stated in appendix B on page 329, each topological sort order of the graph constructed as described in this section indicates a valid one from the perspective of the chosen modeling approach. Given the flexibility of the SEMAT Essence kernel, it is possible to interpret checkpoints differently in a reasonable context-sensitive way, which is potentially leading to another set of interdependencies. Substituting the proposed set of interdependencies below with a different one and following the procedure described in section 3.3 on page 127 would produce a different graph and hence lead to a different set of topological sorting orders defining different possible paths through an SE endeavor.

Interpreting the description of a given Checkpoint might not be without ambiguities. Some of those ambiguities are discussed in the FAQ section of the SEMAT website[248]. While analyzing the description of a Checkpoint each “[...] checklist item should be considered from the context of the alpha it is a part of—taking a different perspective of the same phenomenon. While some checklist items seem closely related and possibly the same as a checklist item in another alpha, when you consider each alpha checklist from its own alpha perspective distinctions often become apparent.”[248]

AVOIDING CIRCULAR DEPENDENCIES To enable the simulation of the Essence kernel in 3.3 on page 127 the proposed way circular dependencies have to be avoided.

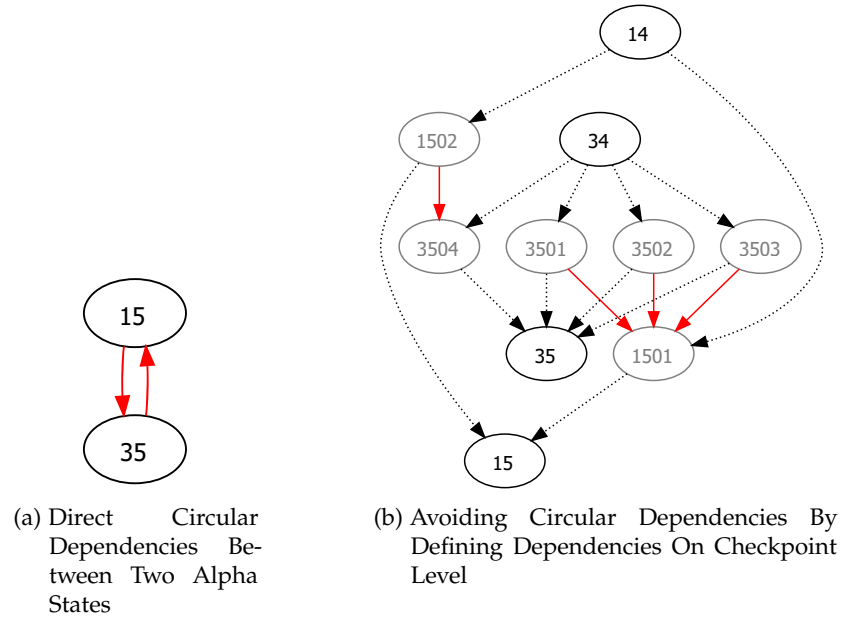


Figure 3.5: Avoiding Circular Dependencies (Direct Dependency)

Some of the Alpha States combine checkpoints with multiple influences from other Alpha States. While it is often sufficient to define one Alpha State as a precondition to another Alpha State, situations occur where one Alpha State *A* depends on another Alpha State *B* which in turn depends directly or indirectly on *A*.

3.5 shows an example where two Alpha States depend on each other and form a cycle from a graph perspective.

Figure 3.6a on page 127 shows another example where two Alpha States depend on each other in an indirect way. Involving other Alpha States this structure forms a cycle from a graph perspective again. In this example, the state *Seeded* (51) of Alpha *Team* is dependent on the state *Bounded* (32) of the Alpha *Requirements*. The state *Principles Established* (71) of Alpha *Way of Working* is dependent on *Team::Seeded* (51) and is itself the predecessor state of *Foundation Established* (72). However, that state is a dependency of *Requirements::Bounded* (32). If dependencies were modeled that way, a deadlock would occur. *Requirements::Bounded* could never get reached because its prerequisite *Way of Working::Foundation Established* itself could never get reached. Defining dependencies on the level of Alpha States does not lead to a valid dependency resolution in that case. To resolve these contradicting dependencies, they have to be defined on a lower level. Figure 3.6b on page 127 shows a solution to the given problem. Not looking to the Alpha State as a whole but on its set of Checkpoints allows a more fine-grained analysis and classification of dependencies. That way a set of Checkpoints was identified (3205, 3206, 3207) that requires *Way of Working::Foundation Established* (72) as a precondition.

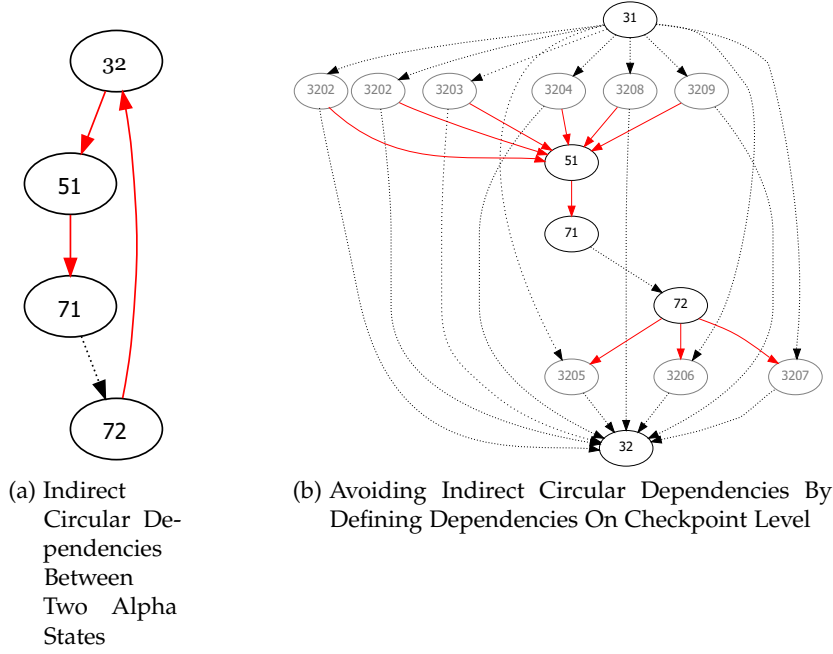


Figure 3.6: Avoiding Circular Dependencies (Indirect Dependency)

The remaining set of Checkpoints (3201, 3202, 3203, 3204, 3208, 3209) were identified as a precondition to *Team::Seeded* (51). The resulting partial graph, shown in figure 3.6b, does not include any cycles.

The example above clarified that it might be required to define dependencies not on Alpha State level but Checkpoint level. Defining the interdependencies on Checkpoints level categorically was not chosen as approach since the high amount of Checkpoints would result in a much more complex model. This gets obvious by inspecting the resulting partial graphs (3.5, 3.6). Therefore the definition of interdependencies on Checkpoint level was only used where necessary to overcome the problem of circular dependencies on Alpha State level. That way it was possible to define interdependencies without any circular dependencies for the whole Essence kernel in a relatively compact model. The result is shown in 3.7. The details of the dependency analysis done on Essence kernel's Alpha States and their Checkpoints is described in detail in Chapter B.

INTER-DEPENDENCY DEFINITION METHOD A first step at defining interdependencies to be included in the simulation model is to identify candidates for interdependencies based on Alpha States. For that purpose, the descriptions of all Checkpoints of all Alpha States of all Alphas have to be semantically analyzed for an indication of an inherent interdependency. Indications for such an inherent interdependency are:

1. the Checkpoint description includes the name of one or more other Alpha(s), e.g. *"The other stakeholders who share the opportunity have been identified."* (of Alpha State Opportunity::Identified),
2. the Checkpoint description does not use the name of one or more other Alphas but a term with same meaning, e.g. *"The mechanisms for managing the requirements are in place."* (of Alpha State Opportunity::Viable)², or
3. the semantics of the Checkpoint description indicate that a precondition has to be met in order to enable a positive assessment of that Checkpoint, e.g. *"A solution has been outlined."* (of Alpha State Opportunity::Viable)³.

These indications have to be analyzed in-depth since they do not automatically represent an interdependency. To give an example: the description of a Checkpoint of State *Identified* of Alpha *Opportunity* is *"An idea for a way of improving current ways of working, increasing market share, or applying a new or innovative software system has been identified."*[188] Here the Alpha names *"software system"* and *"ways of working"* are used, but in this early stage of the endeavor it does not necessarily put any requirements neither on the *Software System* Alpha nor on the *Way of Working* Alpha since that one wasn't even meant at this place.

These examples make clear that this process is not without ambiguities and requires considerable effort. Different from existing simulation approaches utilized in DGBL in SE education this effort requires a modeler to think solely about the underlying domain and to become familiar with ambiguities practitioner may face in their real SE endeavors. This way the modeler becomes at least a more conscious Essence user or at best a better Essence author utilizing insights at creating kernels and practices.

Once all interdependency candidates are identified and captured, they get iteratively added to a dependency graph, capturing all dependencies⁴ and interdependencies. Before the first interdependency candidate gets added to the dependency graph, all dependencies are added.

After that, each interdependency candidate gets added as an edge from the precondition State to the respective dependent State, one by one followed by checking, if the resulting graph is still free of any cycles. Should the added interdependency edge introduce a cycle to

² The *"mechanisms for managing the requirements"* may be read as *Way of Working* Alpha with regards to requirements management.

³ The FAQ section at the SEMAT website states *"A solution can be proposed on the basis of a very sketchy understanding, while the outlined solution suggests that work has been performed to go beyond the initial proposal."*[248]

⁴ representing advancing Alpha States caused by progressing Alphas as well as relationships between Activity Spaces and Alpha States addressed by them

the dependency graph, it has to be removed, and the interdependency has to be defined at the Checkpoint level as described in section 3.3 on page 125.

The iterative process of defining interdependencies, or rather making them explicit, is finished when all interdependency candidates are added as interdependency edges to the dependency graph and that one is still free of any cycles.

Figure 3.7 on the following page shows the result of such a semantic analysis. This (inter-)dependency graph was used in the simulation model utilized for conducting the case study presented in chapter 5 on page 251. Appendix B on page 329 documents the analysis done in detail.

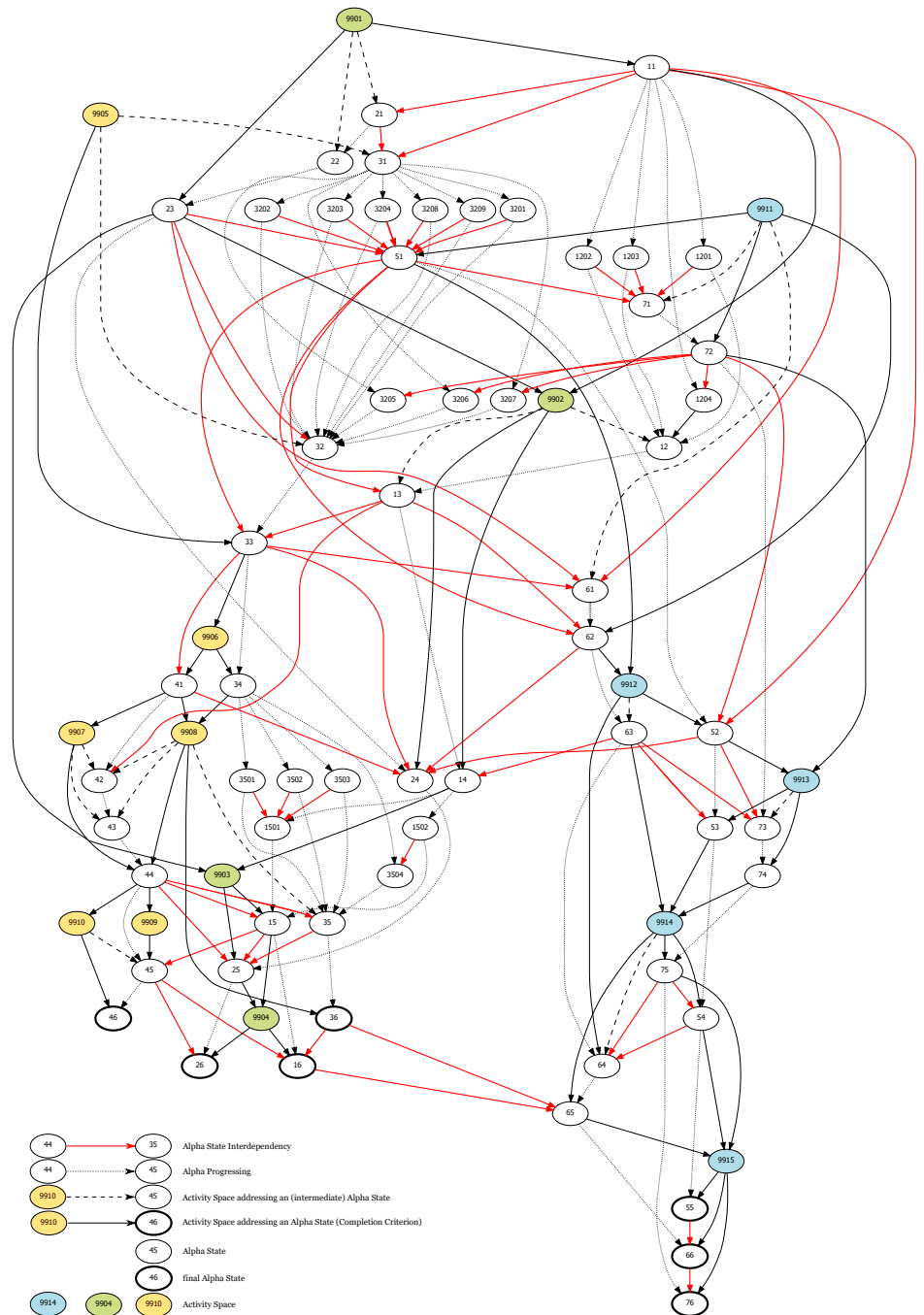
Given the flexibility of the SEMAT Essence Kernel its possible to interpret checkpoints differently in a context-sensitive way potentially leading to another set of (inter-)dependencies. Substituting the proposed set of interdependencies below with different one and following the procedure described in 3.3 would produce a different graph and hence lead to a different set of topological sort orders defining different possible paths through an SE endeavor.

Since Checkpoint descriptions use free text and natural language, it is principally possible that an author is inadvertently introducing Checkpoint descriptions that are not resolvable by the method described above. This may not hinder teams to use the kernel or practice nonetheless in an SE endeavor but represents an ambiguity authors generally will strive to avoid. Using the proposed method such ambiguities caused by circular dependencies could be identified and eliminated.

Requirement: Topological Sortability

To ensure a valid simulation model built with the elements described in section 3.5.3 on page 143, a constructed directed acyclic graph (DAG) $G = (V, E)$ with V defined by the the set of all Alpha States and all their Checkpoints of the chosen Essence kernel/method and E , the set of all edges (u, v) , defined by the set of all implicit and explicit (inter-)dependencies where v is dependent on u 's completion, has to be topological sortable⁵, hence be directed and free of any cycle. A directed edge (u, v) in E indicates that vertex u (representing an AlphaState or Checkpoint) is a precondition to the dependent vertex v

⁵ *Topological Sort*: A topological sort of a directed acyclic graph (DAG) $G = (V, E)$ "is a linear ordering of all its vertices such that if G contains an edge (u, v) , then u appears before v in the ordering. (If the graph contains a cycle, then no linear ordering is possible.)"[63] Algorithms to construct a topological sort of any DAG in linear time include utilizing a Depth First Search (DFS) on G . [63] A topological sort can be performed in $\Theta(V + E)$.



cf. Appendix A on page 325 for the coding scheme of Essence kernel elements used in graphs

Figure 3.7: Alpha State Interdependency Graph Used in the Simulation Model

(again representing an Alpha State or Checkpoint), which means u has to be accomplished before the work on v can be started.

As far as at least one topological sort orders exist, the simulation model is executable in a way that every checkpoint can be fulfilled, hence every Alpha State can be reached, given that each individual effort x , measured in person-hours, associated with one of the Checkpoints in the simulation model is $x \in \mathbb{N}$, $0 \leq x < \infty$.

3.4 DEVS AND FINITE STATE MACHINES

3.4.1 Discrete Event System Specification (DEVS)

The *Discrete Event System Specification (DEVS)* is a formalism with a sound theoretical background. It was introduced in the early 1970s [308]. Since then it was utilized in many different domains and extended over the following decades. The most interesting extensions in the context of this research are *modular, hierarchical models*[309] and *Parallel DEVS*[56]. Both get described in following sections. Besides the modeling formalism, DEVS introduces a framework of abstract simulators [310] building the foundation of many simulation environments and frameworks including the one used for this research.

3.4.2 Classic Atomic DEVS

According to Zeigler et al. [310] a classic atomic (behavioral) discrete event system specification (DEVS) is a structure

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

where

X is the set of *input values*

S is a set of *states*

Y is the set of *output values*

$\delta_{int} : S \rightarrow S$ is the *internal transition function*

$\delta_{ext} : Q \times X \rightarrow S$ is the *external transition function*,

where $Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is the *total state set*,
 e is the *time elapsed* since last transition

$\lambda : S \rightarrow Y$ is the *output function*

$ta : S \rightarrow R_{0,\infty}^+$ is the *time advance function* producing a set of positive reals with 0 and ∞ .

To enable a simulation an initial state $s_0 \in S$ has to be defined.

To make the modeling easier input and output ports are introduced. Keeping the same structure as defined above, the definition of X and

Y changes to

$X = \{(p, v) \mid p \in InPorts, v \in X_p\}$ is the set of input ports and values
 $Y = \{(p, v) \mid p \in OutPorts, v \in Y_p\}$ is the set of output ports and values.

The definition of set X allows for a set of input ports p , each having a defined set of allowed input values X_p . The definition of set Y allows for a set of output ports p , each having a defined set of allowed output values Y_p .

Starting with state $s_0 \in S$ a model may transition between states defined by S triggered by internal or external events. In case of an external event, represented by an incoming message (p, v) at an input port $p \in InPorts$ with value $v \in X_p$, the external transition function δ_{ext} computes the new state $s' \in S$ based on v , the current state $s \in S$ and the elapsed time e for which the model has been in state s . If no external event occurs the model transitions to a new state s' triggered by an internal event after time $ta(s)$ since last external or internal state transition. In that case the new state s' gets determined by the internal transition function δ_{int} based on current state s . The DEVS formalism defines that right before—and only right before—an internal transition the model may generate an output (p, v) with value v and port $p \in OutPorts$ which gets determined by the output function λ applied to the current state s .

To simplify the modeling phase, a variable σ gets introduced to store the time given till next transition and to be considered by the time advance function. The DEVS formalism does not allow an external event to cause an output directly without delay. To enable an output by a transition triggered by an external event, an internal transition would have to be scheduled at same simulation time. This can be accomplished by setting σ to 0 as part of the external transition function δ_{ext} and defining the time advance function as $ta(s, \sigma) = \sigma$.

The following simple example model was chosen to illustrate this modeling approach. The *CostCalculator* model used in the simulation is a simple atomic DEVS model and serves as calculator of incurred costs caused by a virtual team. Costs get calculated on a daily basis omitting weekends. The cost per working day is modeled as constant `COST_PER_WORKING_DAY` and set at model initialization. An incoming message on input port `IN_WORKING_DAY_ENDED` with an arbitrary value triggers the external transition function δ_{ext} which adds the value of `COST_PER_WORKING_DAY` to model's `COST_TOTAL` variable and triggers an immediate output by setting the value of σ to 0, hence triggering an output and internal state transition at same simulation

time. Model's state transition depends solely on external events hence the internal state transition function δ_{int} defines no change of model's state and sets σ to ∞ . Hence the model is waiting for next external event.

$$DEVS_{CostCalculator} = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle,$$

where

$$\begin{aligned} InPorts &= \{ "in" \}, \text{ where } X_{in} = V \text{ (an arbitrary set)} \\ X &= \{ ("in", v) \mid v \in X_{in} \} \text{ is the set of input ports and values} \\ OutPorts &= \{ "out" \}, \text{ where } Y_{out} = M, M \subseteq \mathbb{N}; \forall m \in M : m = \\ &n * cost_per_working_day; n, cost_per_working_day \in \mathbb{N} \\ Y &= \{ ("out", v) \mid v \in Y_{out} \} \text{ is the set of output ports and values.} \\ S &= M \text{ the set of multiples of } COST_PER_WORKING_DAY \\ \delta_{int}(s, \sigma) &= (s, \infty) \mid s \in S \text{ the internal transition function not changing} \\ &\text{state} \\ \delta_{ext}(cost_total, \sigma) &= ((cost_total + cost_per_working_day), 0) \\ \lambda(cost_total) &= cost_total \\ ta(cost_total) = \sigma &= \begin{cases} 0 & \text{if the event was external } (\sigma \text{ set to } 0) \\ \infty & \text{if the event was internal } (\sigma \text{ set to } \infty) \end{cases} \end{aligned}$$

3.4.3 Classic DEVS Coupled Models

The DEVS formalism allows to connect atomic models with ports to form larger structures and more complex models. According to Zeigler et al.[310] a coupled classic DEVS model can be described as

$$N = \langle X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC, Select \rangle,$$

where

$$\begin{aligned} X &= \{ (p, v) \mid p \in InPorts, v \in X_p \} \text{ is the set of input ports} \\ &\text{and values} \\ Y &= \{ (p, v) \mid p \in OutPorts, v \in Y_p \} \text{ is the set of output ports} \\ &\text{and values} \\ D &\text{ is the set of component names} \end{aligned}$$

Component Requirements

Components are DEVS models, for each $d \in D$,

$$M_d = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

is a DEVS with

$$X_d = \{ (p, v) \mid p \in InPorts_d, v \in X_p \}$$

$$Y_d = \{(p, v) \mid p \in OutPorts_d, v \in Y_p\}.$$

Coupling Requirements

External input coupling connects external inputs to component inputs:

$$EIC \subseteq \{((N, ip_N), (d, ip_d)) \mid ip_N \in InPorts, d \in D, ip_d \in InPorts_d\}.$$

External output coupling connects component outputs to external outputs:

$$EOC \subseteq \{((d, op_d), (N, op_N)) \mid op_N \in OutPorts, d \in D, op_d \in OutPorts_d\}.$$

Internal coupling connects component outputs to component inputs:

$$IC \subseteq \{((a, op_a), (b, ip_b)) \mid a, b \in D, op_a \in OutPorts_a, ip_b \in InPorts_b\}.$$

No direct feedback loops are allowed, i.e. not output port of a component may be connected to an input port of the same component:

$$((d, op_d), (e, ip_d)) \in IC \text{ implies } d \neq e.$$

Select : $2^D - \{\}$ $\rightarrow D$ is the tie-breaking function to prioritize between components in case of arbitrary simultaneous events. This function is used in Classic DEVS but eliminated in Parallel DEVS.

3.4.4 *Parallel DEVS*

Parallel DEVS, introduced in 1994[56], differs from classic DEVS in handling imminent components. While the classic DEVS requires to serialize imminent components, Parallel DEVS allows all imminent components to be activated and to send their output to other components. Receiving components are responsible for proper handling of this input. *Messages*, a list of port-value pairs, are the basic exchange medium. Instead of having a single input, Parallel DEVS models allow for a bag of inputs, which may contain multiple occurrences of its elements. To decide the next state of the model in case of colliding simultaneous internal and external events handled by the internal and external transition functions respectively, a third type of transition

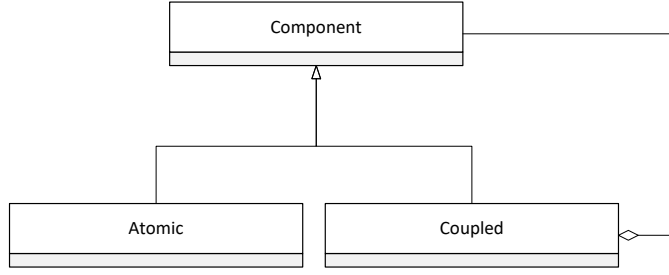


Figure 3.8: DEVS Hierarchical Models

function gets introduced: the *confluent transition*. This function may be used to prioritize the internal and external transition functions.

According to Zeigler et al.[310] a basic Parallel DEVS is a structure,

$$DEVS = \langle X_M, Y_M, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$$

where

$X_M = \{(p, v) \mid p \in InPorts, v \in X_p\}$ is the set of input ports and values

$Y_M = \{(p, v) \mid p \in OutPorts, v \in Y_p\}$ is the set of output ports and values

S is the set of *sequential states*

$\delta_{int} : S \rightarrow S$ is the *internal transition function*

$\delta_{ext} : Q \times X_M^b \rightarrow S$ is the *external transition function*

$\delta_{conf} : Q \times X_M^b \rightarrow S$ is the *confluent transition function*

$\lambda : S \rightarrow Y^b$ is the *output function*

$ta : S \rightarrow R_{0,\infty}^+$ is the *time advance function*

$Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is the *set of total states*.

Parallel DEVS Coupled Models are specified as Classic DEVS except that the *Select* function is omitted. All imminent components generate their outputs which are then distributed to their destination components' input ports based on the coupling information defined on the coupled model.

3.4.5 Hierarchical Models

The Parallel DEVS formalism is closed under coupling[56, 310]. A composite coupled model constructed out of components that are atomic or coupled models is an *Hierarchical Model* and itself a coupled model that could again be integrated into a larger (coupled) model. Figure 3.8 on page 135 illustrates these structural capabilities in an UML [190] class diagram representation. It gets recognizable that this structure conforms to the *Composite* pattern[88].

The simulation model proposed in this thesis is a hierarchical parallel DEVS model with ports. The components represented by parallel atomic and coupled DEVS models, used to construct the hierarchical model dynamically based on a given Essence kernel/method, are introduced in the following section 3.5.

3.4.6 *Finite State Machines (FSM)*

The formalism of *Finite State Machines (FSM)* was utilized where necessary to complement the DEVS formalism at managing the internal state of DEVS models, esp. to overcome the necessarily high number of combined interdependent state variables necessary to represent the inherent complexity of the models. Implemented Finite State Machine (FSM)s get introduced, where implemented DEVS models get described.

3.5 MODEL DESCRIPTION

The simulation model proposed in this thesis is a *deterministic hierarchical parallel DEVS model with ports*. The components, represented by parallel atomic and coupled DEVS models, used to construct the hierarchical model dynamically based on a given simulation model configuration, are introduced in the following subsections.

3.5.1 *Complexity and Level of Detail*

The simulation model is as complex and detailed as the underlying Essence method/kernel that was chosen to build the simulation model.

Considering exclusively the Alphas, their Alpha States, their Checkpoints and the Activities/Activity Spaces addressing those Alpha States of the chosen Essence method or kernel, the chosen Essence method or kernel itself defines the basic elements of the simulation model. Adding a minimum set of supplemental concepts and model elements, this approach ensures highly transparent simulation models without “hidden surprises”.

Simulating the Essence kernel means in this context, that the virtual workforce is represented as a whole by the *Alpha Team*. If a modeler would like to represent single team members, an additional (sub-)Alpha *Team Member* with all its Alpha States, their Checkpoints, and one or more Activities addressing these Alpha States of the new Alpha, would have to be defined by an Essence practice that was included in the underlying Essence method⁶. The same applies for any

⁶ Currently the simulation model supports only elements of the SEMAT Essence kernel (cf. section 3.5.5 on page 169).

other element that gets represented by an Alpha, e.g. *bugs, requirements items*, etc. That way the simulated model is as detailed as the underlying Essence method/kernel, presenting the same amount of information and support as the Essence method/kernel does in a real SE endeavor.

3.5.2 *Hierarchical Parallel DEVS Model With Ports*

The introduced simulation model is a hierarchical parallel DEVS model with ports. The model for the Essence kernel can be described as

$$\begin{aligned}
N &= \langle X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC \rangle, \text{ where} \\
InPorts &= \{ "in_resume", "in_resource_assignment" \} \\
X_{in_resume} &= \{ "resume" \} \\
X_{in_resource_assignment} &= \{ (d_{a1}, v_1), \dots, (d_{az}, v_z) \mid d_{a1..az} \in D_{ActivitySpace}, \\
&\quad v_{1..z} \in \mathbb{N}, \sum_i v_i \leq RESOURCES_AVAILABLE \} \\
OutPorts &= \{ "out_ui_message" \} \\
Y_{out_ui_message} &= V(\text{an arbitrary set}) \\
D &= \{ ActivitySpace, Alpha, AlphaState, Checkpoint, \\
&\quad MultiActivityCheckpoint, CalendarTime, \\
&\quad CostCalculator, AssignmentAllocator, SimPause, \\
&\quad UIHttpSender, SinglePerformanceFactor, \\
&\quad TotalPerformanceFactor, IntelligentTutor, \\
&\quad SimulationReporter \} \\
M_{activity_space_{1..15}} &= ActivitySpace \\
M_{alpha_{1..7}} &= Alpha \\
M_{alpha_state_{1..41}} &= AlphaState \\
M_{checkpoint_{1..i}} &= Checkpoint \\
M_{multi_activity_checkpoint_{1..j}} &= MultiActivityCheckpoint \\
M_{calendar_time} &= CalendarTime \\
M_{cost} &= CostCalculator \\
M_{assignment_allocator} &= AssignmentAllocator \\
M_{sim_pause} &= SimPause \\
M_{performance_factor} &= SinglePerformanceFactor \\
M_{total_performance_factor} &= TotalPerformanceFactor \\
M_{ui_http_sender} &= UIHttpSender \\
M_{sim_reporter} &= SimulationReporter \\
M_{sim_tutor} &= IntelligentTutor \\
&\quad \{ ((N, "in_resume"), ("sim_pause", "in_resume")), \\
EIC &= ((N, "in_resource_assignment"), \\
&\quad (assignment_allocator, "in_resource_assignment")) \} \\
EOC &= \{ ((ui_http_sender, "out_ui_message"), \\
&\quad (N, "out_ui_message")), \\
&\quad ((ui_http_sender, "out_sim_report"), \\
&\quad (N, "out_sim_report")) \} \\
IC &= IC_{cal_time} \cup IC_{pause} \cup IC_{perf} \cup IC_{report} \cup IC_{precond} \cup \\
&\quad IC_{activity_cp} \cup IC_{alloc} \cup IC_{alpha_state} \cup IC_{state_cp}
\end{aligned}$$

The Internal Couplings IC are assembled from the following partial sets:

- The calendar time internal couplings IC_{cal_time} connecting the CalendarTime DEVS model with all models in need of information about start and end of (working) days. The CalendarTime DEVS model is described in section 3.5.3 on page 157.

$$IC_{cal_time} = \{((calendar_time, "out_end_working_day"), (sim_tutor, "in_working_day_ended")), ((calendar_time, "out_end_working_day"), (cost, "in_working_day_ended")), ((calendar_time, "out_end_working_day"), (checkpoint_{1..i}, "in_working_day_stop")), ((calendar_time, "out_end_working_day"), (activity_space_{1..15}, "in_working_day_stop")), ((calendar_time, "out_start_working_day"), (checkpoint_{1..i}, "in_working_day_start")), ((calendar_time, "out_start_working_day"), (activity_space_{1..15}, "in_working_day_stop"))\}, \text{ where}$$

$checkpoint_{1..i}$ is the set of Checkpoint DEVS model instances including the set of MultiActivityCheckpoint DEVS model instances and $activity_space_{1..15}$, the set of 15 ActivitySpace DEVS model instances.

- The simulation reporter internal couplings IC_{report} connecting DEVS models of relevance with the SimulationReporter model collecting simulation state information.

$$\begin{aligned}
IC_{report} = \{ & ((checkpoint_{1..i}, "out_status"), \\
& (sim_reporter, "in_status")), \\
& ((calendar_time, "out_new_day_started"), \\
& (sim_reporter, "in_status")), \\
& ((calendar_time, "out_status"), \\
& (sim_reporter, "in_status")), \\
& ((cost, "out_money_spent"), \\
& (sim_reporter, "in_status")), \\
& ((resource_allocator, "out_status"), \\
& (sim_reporter, "in_status")), \\
& ((total_performance_factor, out_status), \\
& (sim_reporter, "in_status")), \\
& ((performance_factor, out_status), \\
& (sim_reporter, "in_status")), \\
& ((activity_space_{1..15}, out_status), \\
& (sim_reporter, "in_status")), \\
& ((alpha_{1..7}, out_status), \\
& (sim_reporter, "in_status")), \\
& ((sim_reporter, "out_status"), \\
& (ui_http_sender, "in_sim_report")) \}, \text{ where}
\end{aligned}$$

- The pause internal couplings IC_{pause} enabling interactivity in terms of input and output of the simulation model by connecting Checkpoint and IntelligentTutor with the SimulationPause, ResourceAllocator and UiHttpSender models.

$$\begin{aligned}
IC_{pause} = \{ & ((sim_tutor, "out_tutoring_ui_message"), \\
& (sim_pause, "in_pause_simulation")), \\
& ((sim_tutor, "out_tutoring_ui_message"), \\
& (ui_http_sender, "in_ui_message")), \\
& ((sim_tutor, "out_tutoring_ui_message"), \\
& (resource_allocator, "in_awaiting_assignment")), \\
& ((checkpoint_{1..i}, "out_ui_message"), \\
& (sim_pause, "in_pause_simulation")), \\
& ((checkpoint_{1..i}, "out_ui_message"), \\
& (ui_http_sender, "in_ui_message")) \}, \text{ where}
\end{aligned}$$

$checkpoint_{1..i}$ is the set of Checkpoint DEVS model instances including the set of MultiActivityCheckpoint DEVS model instances.

- The performance couplings IC_{perf} enabling a representation of team performance in the simulation model:

$$IC_{perf} = \{((\alpha_{1..p}, "out_status"), (performance_factor_{1..p}, "in_alpha_status")), ((performance_factor_{1..p}, "out_current_factor"), (total_performance_factor, "in_single_performance_factor")), ((total_performance_factor, "out_total_performance_factor"), (checkpoint_{1..i}, "in_effective_working_power_factor"))\}, \text{ where}$$

each α_i that got defined as performance factor gets coupled to its exclusive corresponding $performance_factor_i$ and all of these $performance_factor_{1..p}$ get coupled to the one $total_performance_factor$ which aggregates them and is in turn coupled to each of the Checkpoints $checkpoint_{1..i}$ including all MultiActivityCheckpoint instances.

- The Checkpoint/ActivitySpace coupling connecting all ActivitySpace instances bidirectionally with all those Checkpoint instances that they are addressing, enabling the assignment of resources, their loss due to inactivity in the case of an ongoing activity, and the signaling of fulfillment of a Checkpoint:

$$IC_{activity_cp} = \{((activity_space_i, "out_checkpoint_{i,j_resources_assigned"}), (checkpoint_{i,j}, "in_resources_assigned")), ((activity_space_i, "out_checkpoint_{i,j_lost}"), (checkpoint_{i,j}, "in_lost")), ((checkpoint_{i,j}, "out_fulfilled"), (activity_space_i, "in_checkpoint_completed"))\}, \text{ where}$$

$checkpoint_{i,j}$ represents a Checkpoint j that gets addressed by an ActivitySpace $activity_space_i$ and $"out_checkpoint_{i,j_resources_assigned}"$ as well as $out_checkpoint_{i,j_lost}$ represent the set of output ports, where each port is exclusively coupled to one $checkpoint_{i,j}$.

- The Alpha/Alpha State couplings connecting all Alpha instances bidirectionally with their respective set of Alpha State instances enabling the signaling of reached and lost (predecessor) states:

$$IC_{\alpha_state} = \{((\alpha_i, "out_alpha_state_{ij_predecessors_reached}"), (\alpha_state_{ij}, "in_predecessor_states_reached")), ((\alpha_i, "out_alpha_state_{ij_predecessors_lost}"), (\alpha_state_{ij}, "in_predecessor_states_lost")), ((\alpha_state_{ij}, "out_fulfilled"), (\alpha_i, "in_state_reached")), ((\alpha_state_{ij}, "out_state_lost"), (\alpha_i, "in_state_lost"))\}, \text{ where}$$

α_state_{ij} represents an Alpha State instance j that is part of an Alpha instance α_i and $"out_alpha_state_{ij_predecessors_reached}"$ as well as $out_alpha_state_{ij_predecessors_lost}$ represent a set of output ports, where each port is exclusively coupled to one α_state_{ij} .

- The Alpha State/Checkpoint couplings connecting each of the Alpha State instances bidirectionally with their respective set of Checkpoint instances enabling the signaling of satisfied preconditions and fulfilled or lost Checkpoints:

$$IC_{state_cp} = \{((\alpha_state_i, "out_checkpoints_preconditions_satisfied"), (checkpoint_{ij}, "in_alpha_state_preconditions_satisfied")), ((checkpoint_{ij}, "out_fulfilled"), (\alpha_state_i, "in_checkpoint_fulfilled")), ((checkpoint_{ij}, "out_lost"), (\alpha_state_i, "in_checkpoint_fulfilled"))\}, \text{ where}$$

$checkpoint_{ij}$ represents an Checkpoint instance j contained by an Alpha State instance i .

- The precondition couplings, representing the preconditions defined explicitly by the modeler. Precondition couplings connect each of the model instances, serving as precondition and represented by an Alpha State or Checkpoint model instance, with each of its dependent models, which may be a set made out of any combination of Alpha States and/or Checkpoints:

$$IC_{precond} = \{((precondition_i, "out_fulfilled"), (dependent_{ij}, "in_preconditions_satisfied"))\}, \text{ where}$$

$precondition_i$ represents an Alpha State or Checkpoint model instance i which was defined as precondition to all of $dependent_{ij}$, each representing an other Alpha State or Checkpoint j ($i \neq j$) dependent on i .

- The resource allocation couplings connecting the ResourceAllocator model instance with all of the ActivitySpace instances via an exclusive output port for each of the ActivitySpaces:

$$IC_{alloc} = \{((resource_allocator, "out_activity_i"), (activity_space_i, "in_resource_assignment"))\}.$$

The DEVS models coupled internally and externally get described in the next section. A rigorous formal description as done for the *CostCalculator* atomic DEVS model in section 3.4.2 on page 131 was omitted for the sake of brevity. All the models explained in following subsections follow the definition of atomic parallel DEVS with ports described in section 3.4.4 on page 134, except for the *MultiActivity-Checkpoint* model which represents a coupled parallel DEVS model. Each of the models gets explained with its motivation, its most important characteristics, its role in the system and its input/output capabilities.

3.5.3 Model Elements Overview

The following sections provide an overview and descriptions of the model's elements. Table 3.1 on the following page provides a list of the included models.

Model name	Location
ActivitySpace	Section 3.5.3
Alpha	Section 3.5.3 on page 146
AlphaState	Section 3.5.3 on page 148
Checkpoint	Section 3.5.3 on page 149
MultiActivityCheckpoint	Section 3.5.3 on page 153
CalendarTime	Section 3.5.3 on page 157
CostCalculator	Section 3.5.3 on page 161
AssignmentAllocator	Section 3.5.3 on page 161
SimPause	Section 3.5.3 on page 162
SinglePerformanceFactor	Section 3.5.3 on page 163
TotalPerformanceFactor	Section 3.5.3 on page 163
IntelligentTutor	Section 3.5.3 on page 165
UIHttpSender	Section 3.5.3 on page 165
SimulationReporter	Section 3.5.3 on page 165

Table 3.1: List of Models

ActivitySpace

An ActivitySpace in the simulation model represents an Essence Activity Space. Essence Activity Spaces are generic abstract placeholders for concrete Activities that may get defined by Essence practices (cf. section 2.7 on page 95). To enable the simulation of the Essence kernel, where no concrete Activities are defined, Activity Spaces are handled in the same way as concrete Activities⁷ in the context of the simulation model.

As described in section 2.7.2 on page 103, Activity Spaces target at reaching defined Alpha States, each assessable by a set of Checkpoints. The simulation model manifests this concept by sequentially fulfilling the Checkpoints of the addressed Alpha State(s). To simplify the simulation model, the virtual workforce processes Checkpoints in a sequential style. The sequence in which Checkpoints of addressed Alpha States get fulfilled—conceptionally the result demanded by a Checkpoint gets achieved—is defined by the modeler. At this, the mixing of Checkpoints of different Alpha States is allowed, as long as Checkpoints belonging to different Alpha States of one Alpha are not

⁷ In the Essence language both, Activity and ActivitySpace share the same base class AbstractActivity.

mixed. Checkpoints of one Alpha have to be sequenced in the order defined by the sequence of Alpha States they belong to.

To perform an Activity consumes resources—to achieve the result demanded by a Checkpoint needs effort to be done. From a DEVS perspective, an Activity serves as resource consuming service station. The number of activities and the effort done by the virtual team is limited by the person-hours per working day available, a parameter of the simulation model. The simulation model allows to split the available person-hours per working day and to allocate the amount to a different set of activities for the desired period. Once the virtual workforce spent enough effort performing an Activity a Checkpoint of the addressed Alpha State gets fulfilled given all preconditions for working on that Checkpoint are satisfied beforehand (cf. 3.5.3 and 3.5.3). Performing—or better non-performing—an Activity, where not all preconditions are satisfied, consumes resources too but without reaching the addressed Alpha State by fulfilling its describing Checkpoints.

An Activity may have an ongoing character, which means that while performing it once may result in a targeted Alpha State, that state could get lost afterward, if the Activity would not be performed anymore. Some Activities of the ENDEAVOR Area of Concern may serve as examples. Once a team would not be supported anymore (SUPPORT THE TEAM), once a team would stop to track its progress (TRACK PROGRESS) or stop to coordinate its activities (COORDINATE ACTIVITY) its performance would certainly decrease. Because this is an important concept, it is represented in the simulation model. A modeler may define a minimum amount of person-hours per working day that would have to be performed to maintain a state once reached. If the virtual workforce were not assigned with at least that minimum amount to the activity, checkpoints would not be fulfilled anymore. They would get lost resulting in lost Alpha States after a defined number of working days without performing the ongoing activity. The sequence and number of working days in which Checkpoints once fulfilled by performing a given Activity would get lost are defined as parameters of the model. The same applies to the number of person-hours that would have to be performed to regain those Checkpoints and hence the corresponding Alpha State. If the amount of assigned person-hours per working day is not sufficient (lower as the minimum amount defined by the modeler as a parameter), no progress is made by the virtual workforce at performing this activity.

Table 3.2 on page 146 describes the input and output ports of the ActivitySpace DEVS atomic model. The last column of the table shows the quantity of the respective DEVS port. Most of the ports are created just once per ActivitySpace instance others are dynamically

Type	Name	Purpose	Quantity
IN	in_resource_assignment	receive assigned resources	1
IN	in_checkpoint_completed	get informed about completed Checkpoints	1
IN	in_working_day_start	get informed about start and end of a working day	1
IN	in_working_day_stop		1
OUT	out_checkpoint_<X>_resources_assigned	inform current checkpoint about resource assignment	dynamic: 1 per addressed Checkpoint
OUT	out_checkpoint_<Y>_lost	inform checkpoints "to maintain" about lost status	dynamic: 1 per Checkpoint "to maintain"
OUT	out_status	measurement and intelligent tutoring	1

Table 3.2: I/O Ports of ActivitySpace DEVS Model

determined at model creation time to reflect a number of associated atomic DEVS models of the respective type.

Figure 3.9 on page 147 shows the state machine implemented to manage the state of the ActivitySpace model. An ActivitySpace starts in state *WAITING*. Once resources get assigned, it gets in state *PROGRESSING* and assigns its resources to the first Checkpoint in its list of addressed Checkpoints. Should the amount of resources assigned be too low it gets in a *NOT-PROGRESSING* state where no progress is made. Once all Checkpoints are fulfilled the ActivitySpace gets in the final state *COMPLETED*, if it is not representing an ongoing activity. In case it should represent an ongoing activity, it gets in state *MAINTENANCE*. Once in that state, it may lose checkpoints due to insufficient resources assigned to it and transition to state *RETROGRESSING*, where it may lose more and more of the once fulfilled Checkpoints. If enough resources get assigned to the ActivitySpace, it gets in state *PROGRESSING* resulting in fulfilled Checkpoints and eventually in state *MAINTENANCE* again, once all Checkpoints got fulfilled.

In a simulation model of the Essence kernel [188] 15 atomic DEVS models of this type ActivitySpace exist.

Alpha

As the ActivitySpace atomic DEVS model, the Alpha atomic DEVS model represents the Essence element type of the same name. An Alpha DEVS model manages its Alpha States by being informed about reached and lost Alpha States it is containing. At the same time the Alpha DEVS model informs subsequent Alpha States about the accomplishment and loss of their respective predecessor states.

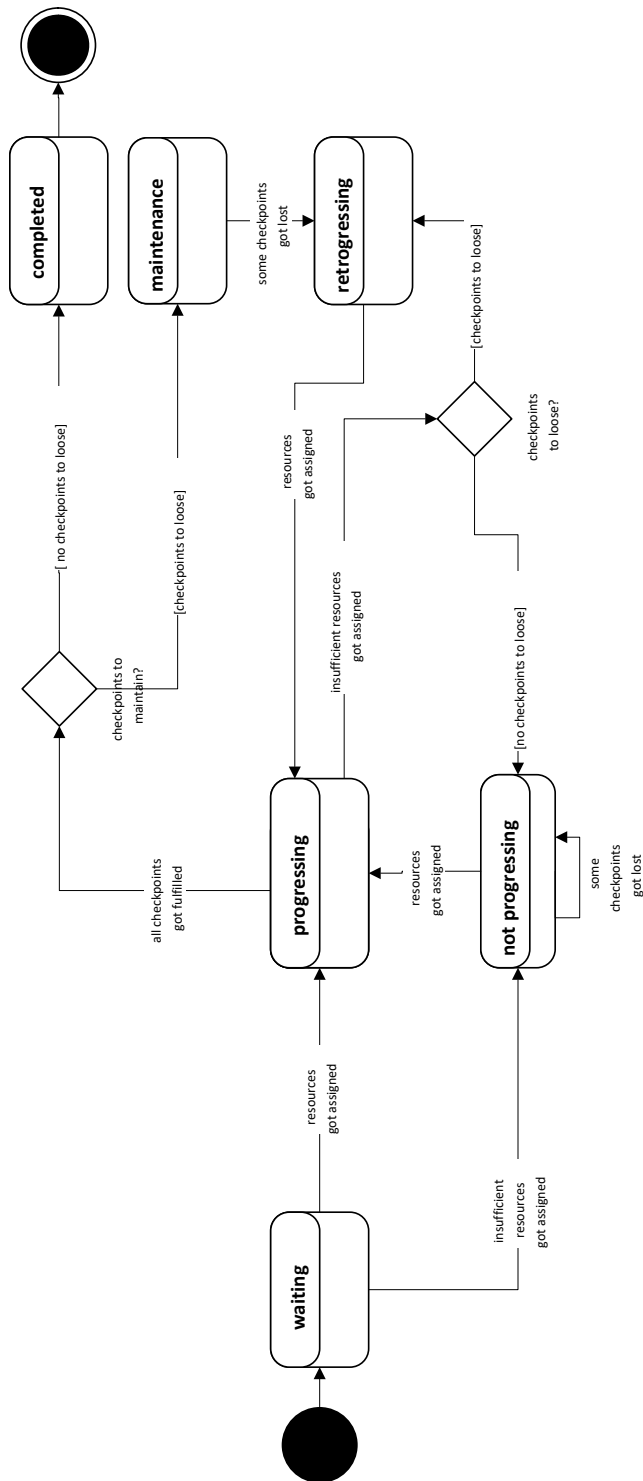


Figure 3.9: FSM ActivitySpace



Figure 3.10: I/O Ports of Alpha DEVS Model

Type	Name	Purpose	Quantity
IN	in_state_reached	get informed about the accomplishment of a contained Alpha State	1
IN	in_state_lost	get informed about the loss of a contained Alpha State (cf. 3.5.3)	1
OUT	out_alpha_state_<X> _predecessors_reached	inform subsequent Alpha State about the accomplishment of preceding state	dynamic: 1 per contained Alpha State
OUT	out_alpha_state_<Y> _predecessors_lost	inform subsequent Alpha State about the loss of preceding state	dynamic: 1 per contained Alpha State
OUT	out_status	measurement and intelligent tutoring	1

Table 3.3: I/O Ports of Alpha DEVS Model

Figure 3.10 on page 148 shows the input and output ports of the DEVS model. Table 3.3 on page 148 describes these ports and gives information about their purpose and quantity. Some of the ports are created just once per Alpha instance. The number of others gets determined dynamically at model creation time to reflect the number of associated Alpha States. In a simulation model of the Essence kernel[188], 7 atomic DEVS models of type Alpha exist.

AlphaState

The AlphaState atomic DEVS model represents an Essence Alpha State. It keeps track of the state of its own preconditions, of the accomplishment of its predecessor AlphaState, and manages its Checkpoints by observing their state and keeping them informed about the fulfillment of its preconditions. AlphaState informs its containing Alpha about its changes in state. Figure 3.11 on page 149 shows the input and output ports of the DEVS model. Table 3.4 on page 150 describes these ports and gives information about their purpose and quantity.

Both, the state of the preceding AlphaState and the preconditions in form of interdependencies explicitly defined by the modeler, represent preconditions to an AlphaState. These different types are handled in separate ways to clearly distinguish between the former type that is defined by the Essence kernel and language and the latter type that represents an addition of this simulation approach (cf. 3.3).

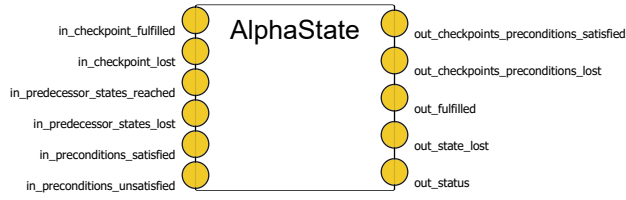


Figure 3.11: I/O Ports of AlphaState DEVS Model

Figure 3.12 on page 151 visualizes the inner states of the atomic DEVS model and their transitions defined by an FSM. In the simplest case an AlphaState instance has no defined preconditions and is waiting solely for the accomplishment of its predecessor state. If that state was reached, the AlphaState signals this accomplishment to its checkpoints, gets in state *RUNNING* and keeps track of its fulfilled checkpoints. Once all its checkpoints got fulfilled, the AlphaState is *REACHED* and signals its containing Alpha its own accomplishment.

As described in 3.3 and 3.3 things can get more complex. If a modeler defined preconditions of the AlphaState instance in the form of interdependencies, all of its preconditions would have to be satisfied before the AlphaState gets in state *RUNNING*. After an AlphaState got in state *REACHED*, one of its own Checkpoints, one of its predecessor's ones, or any of the defined preconditions may get lost, resulting in a state transition to one of the *LOST...* states. In order to transition to the *REACHED* state again, all of its own Checkpoints would have to be fulfilled, the predecessor AlphaState would have to be *REACHED*, and all preconditions (Alpha States of different Alphas or single Checkpoints of them) would have to be satisfied again. Figure 3.12 on page 151 visualizes the inherent complexity implemented in a FSM.

In a simulation model of the Essence kernel[188], 41 atomic DEVS model instances of type AlphaState exist.

Checkpoint

This atomic DEVS model represents an Essence Checkpoint, more precisely an Essence Checkpoint getting targeted by a single Activity Space. Activities/Activity Spaces address Alpha States (cf. section 2.7.2 on page 103). Performing an Activity(Space) leads to the results required by the Checkpoints contained by Alpha States. In this context, a Checkpoint is considered to represent a task leading to the result that is described by the Checkpoint itself. Accomplishing that task consumes resources that have to be assigned to the an Activity(Space) that is addressing the AlphaState containing the Checkpoint, and hence addressing the Checkpoint itself.

Before any progress towards the results required by a Checkpoint can be made, all of its preconditions have to be satisfied. This is a requirement of the simulation model assumed to keep it simple enough, that might be more rigorous than in reality. Preconditions of

Type	Name	Purpose	Quantity
IN	in_checkpoint_fulfilled	get informed about the accomplishment of a contained Alpha State	1
IN	in_checkpoint_lost	get informed about the loss of a contained Alpha State (cf. 3.5.3)	1
IN	in_predecessor_states_reached	get informed about the accomplishment of the preceding state(s)	1
IN	in_predecessor_states_lost	get informed about the loss of the preceding state(s)	1
IN	in_preconditions_satisfied	get informed about the accomplishment of a precondition	1
IN	in_preconditions_unsatisfied	get informed about the loss of a precondition	1
OUT	out_checkpoints_preconditions_satisfied	inform all contained Checkpoints about accomplishment of precondition	1
OUT	out_checkpoints_preconditions_lost	inform all contained Checkpoints about loss of precondition	1
OUT	out_fulfilled	inform containing Alpha about the accomplishment of state itself	1
OUT	out_state_lost	inform containing Alpha about the loss of state itself	1
OUT	out_status	measurement and intelligent tutoring	1

Table 3.4: I/O Ports of AlphaState DEVS Model

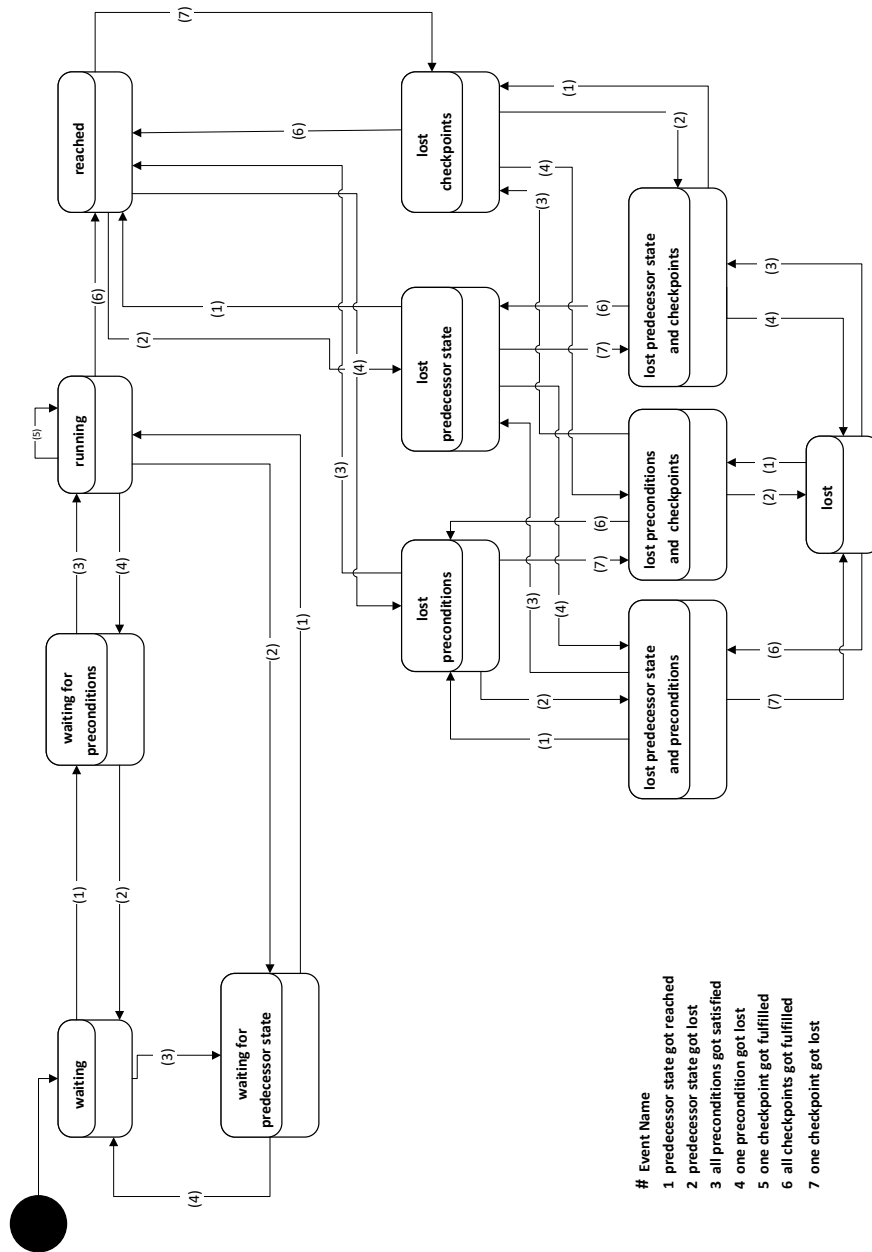


Figure 3.12: FSM Alpha State

a Checkpoint are composed of preconditions of the AlphaState that contains the Checkpoint and any preconditions defined by the modeler on the Checkpoint itself (cf. 3.3). For that reason, a Checkpoint holds a set of preconditions and tracks their satisfaction by getting informed about their accomplishment. On the other hand, the same Checkpoint might be a precondition to other dependent model elements (AlphaState(s) and/or other Checkpoint(s)). In that case, it will inform all dependent elements about its own accomplishment by sending a message via the respective output port (cf. *out_fulfilled* in Table 3.6 on page 155).

Figure 3.14 on page 156 shows the state machine implemented to represent the inner state of a Checkpoint. A Checkpoint starts in state *WAITING*. It transitions to state *RUNNING* through a set of *WAITING FOR ...* states by getting resources assigned and getting all of its own and all of its AlphaState's preconditions satisfied. Once all required effort is done, it transitions to *FULFILLED* state. In case of a Checkpoint getting addressed by an ActivitySpace, representing work that needs performed in an ongoing fashion, it might transition to *LOST* state due to inactivity. From there, it transitions to state *REGAINING*, when resources get assigned and to *FULFILLED* once again, when all effort required to regain got done.

Table 3.6 on page 155 summarizes the input and output ports of the atomic DEVS model illustrated in Figure 3.13 on page 153 and describes their quantity and purpose.

Resources might be assigned to and revoked from an ActivitySpace, hence a Checkpoint, multiple times in any time period. The progress made by fulfilling a Checkpoint, is affected by the amount of resources, measured in person-hours per working day, and the current performance of the virtual workforce, expressed by an effective working power factor, which aggregates different influences on workforce's performance (cf. 3.5.3). The effort to be done to achieve the results required by the Checkpoint, is given by the modeler and measured in person-hours.

The effort done on a Checkpoint in a time period $(t_j - t_i)$ where *EffectiveWorkingPowerFactor* (EWPF) and *PersonsAssigned* (PA) are constant is defined by

$$EffortDone_{t_i, t_j} = \text{Minimum}(EffortRequired, (t_j - t_i) * PA_{t_i} * EWPF_{t_i}), \text{ where}$$

$$t_i \leq t_j, PersonsAssigned_{t_i} = PersonsAssigned_{t_j},$$

$$PA_t = \frac{AssignedPersonHoursPerWorkingDay_t}{WorkingHoursPerWorkingDay},$$

$$EWPF_{t_i} = EWPF_{t_j}.$$

The cumulated *EffortDone* at point in time t_n is

$$EffortDone_{t_n} = \sum_{i=1}^n EffortDone_{t_i, t_j}.$$

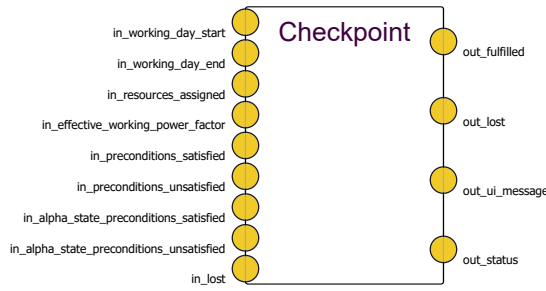


Figure 3.13: I/O Ports of Checkpoint DEVS Model

Changes of resource allocation and of the effective working power factor over time are considered by the model. At every change of factors of the effort calculation, caused by an external event, the *time advance* function ta schedules an internal event at Checkpoint's newly calculated completion time. If no further external event interferes the *internal transition function* δ_{int} gets invoked and ensures the transition to state FULFILLED via an EFFORT_GOT_DONE event at Checkpoint's state machine, once all effort required got done.

The Checkpoint model provides feedback to the user of the simulation via messages at the *out_ui_message* output port enabling the trace of events and empowering an interactive environment. To provide meaningful feedback, messages for all relevant events have to be defined as part of the modeling process (cf. 3.5.3). Table 3.5 on page 154 summarizes the types of messages sent by a Checkpoint at certain events and gives examples of such messages.

In a simulation model of the Essence kernel[188], 183 atomic DEVS model instances of type Checkpoint exist. 24 Checkpoints get addressed by more than one ActivitySpace. Those special Checkpoints are represented by a coupled DEVS model called MultiActivityCheckpoint that gets described in next section.

MultiActivityCheckpoint—Checkpoints of Alpha States with Multiple Targeting Activities

In the Essence kernel[188], Alpha States are addressed by ActivitySpaces defining them as completion criteria. Each Alpha State has a list of Checkpoints enabling a consistent assessment of the Alpha State. An Alpha State is reached only, if all of its Checkpoints are fulfilled. Performing an Activity(Space) leads to the results required by the Checkpoints of an Alpha State. Results described by a Checkpoint may get achieved by performing just one Activity(Space) or may require to perform multiple Activities (cf. Figure 3.4b on page 124).

This coupled DEVS model represents an Essence Checkpoint which is part of an Alpha State that gets addressed by multiple Activities. The modeling approach of the simulation model allows to specify if the Checkpoint of such an AlphaState gets addressed by just one

Message Type	Purpose	Example
STARTED	signal start of work performed on this Checkpoint (sent when resources got assigned by ActivitySpace addressing the Checkpoint via its AlphaState)	<i>"We start to define responsibilities of the stakeholder representatives."</i>
STOPPED	signal stop of work performed on this Checkpoint (sent when resources got revoked by ActivitySpace addressing the Checkpoint via its AlphaState)	<i>"The stakeholder representatives stopped to get authorization to carry out their responsibilities."</i>
FULFILLED	signal fulfillment of this Checkpoint (sent when the required effort measured in person-hours had been performed)	<i>"All stakeholder representatives are now authorized to carry out their responsibilities."</i>
LOST	signal loss of Checkpoint due to inactivity (sent when no resources were assigned to an ActivitySpace representing ongoing work for a given period of time), applies only if Checkpoint was fulfilled before and if ActivitySpace has an ongoing character	<i>"We lost the ability to bring unplanned work under control."</i>
STARTED REGAINING	signal start of work performed on a lost Checkpoint (sent when resources got reassigned by ActivitySpace addressing the Checkpoint via its AlphaState)	<i>"We start to regain the ability to bring unplanned work under control."</i>
REGAINED	signal (re-)fulfillment of once lost Checkpoint (sent when the required effort measured in person-hours to regain the Checkpoint had been performed)	<i>"We regained the ability to bring unplanned work under control."</i>

Table 3.5: Messages sent by Checkpoint DEVS Model to Provide Feedback in an Interactive Environment

Type	Name	Purpose	Quantity
IN	in_working_day_start	signal start and end of a working day to enable calculation of effort done	1
IN	in_working_day_stop		1
IN	in_resources_assigned	get the current resources assigned (expressed in person-hours per working day) to achieve the results demanded by the description of this checkpoint	1
IN	in_effective_working_power_factor	get the current effective working power factor, reflecting the current performance of the virtual workforce (cf. 3.5.3, 3.5.3)	1
IN	in_preconditions_satisfied	get informed about the accomplishment of a precondition, allowing progress if all preconditions are satisfied	1
IN	in_preconditions_unsatisfied	get informed about the loss of a precondition, no progress possible	1
IN	in_alpha_state_preconditions_satisfied	get informed about the accomplishment of a precondition of containing AlphaState, allowing progress	1
IN	in_alpha_state_preconditions_unsatisfied	get informed about the loss of a precondition of containing AlphaState, no progress possible	1
IN	in_lost	get informed that this Checkpoint got lost due to insufficient resources applied to addressing Activity(Space), applies only if Checkpoint was fulfilled before and if ActivitySpace represents ongoing activities	1
OUT	out_fulfilled	inform containing AlphaState, addressing ActivitySpace and any potential dependent element about the fulfillment of this Checkpoint	1
OUT	out_state_lost	inform containing AlphaState about the loss of this Checkpoint	1
OUT	out_ui_message	enable feedback and interactive use of the simulation model	1
OUT	out_status	measurement and intelligent tutoring	1

Table 3.6: I/O Ports of Checkpoint DEVS Model

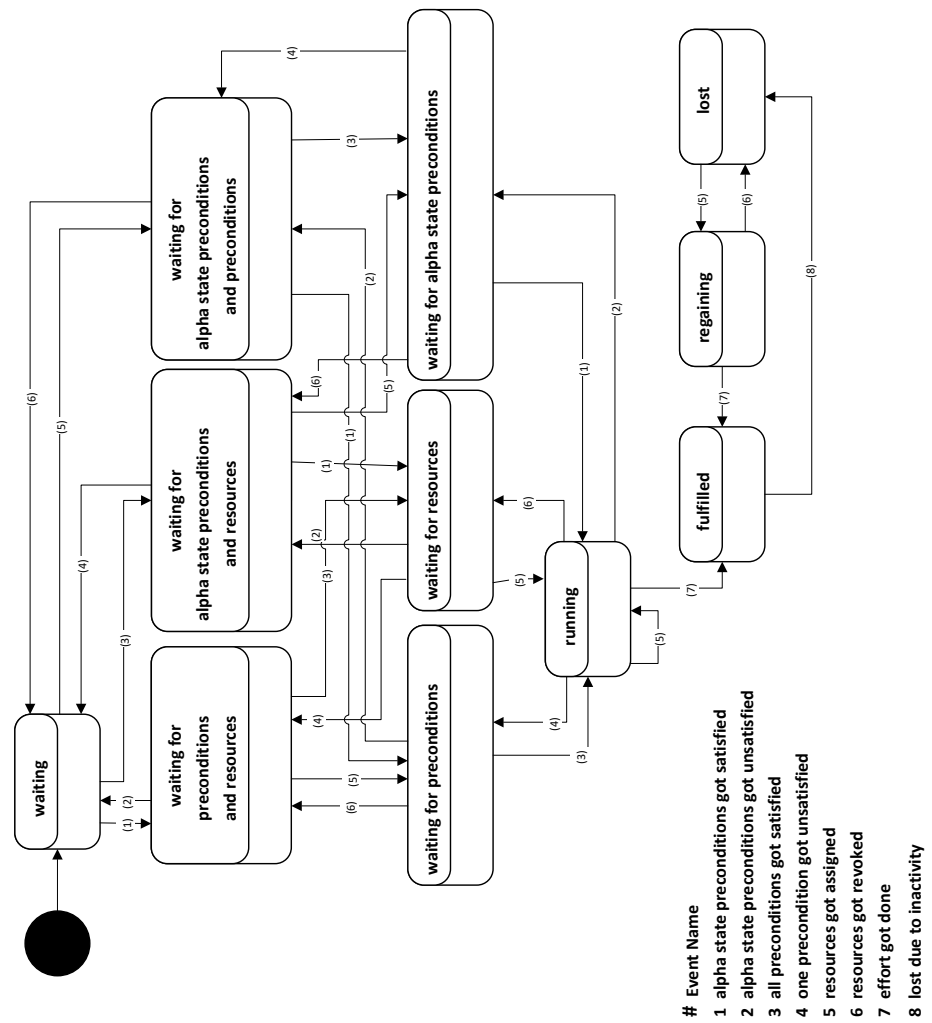


Figure 3.14: FSM Checkpoint DEVS Model

Activity(Space) (cf. 3.5.3) or by multiple ones, which gets described here.

This type of DEVS model is a special one. While all other introduced DEVS models are atomic models this one is a coupled DEVS model reusing the atomic Checkpoint model (cf. 3.5.3) by decorating that one with functionality to consider multiple addressing Activity(Spaces). This coupled model shows the flexibility of the chosen DEVS simulation formalism.

The coupled DEVS model consists of three types of atomic DEVS models and the respective internal and external input/output couplings. These elements are depicted in Figure 3.15 on page 161. Each coupled MultiActivityCheckpoint model contains one atomic model MULTIACTIVITYCHECKPOINTALLOCATOR, one atomic model MULTIACTIVITYCHECKPOINTAGGREGATOR and a number of atomic CHECKPOINT models reflecting the number of Activities addressing the Checkpoint.

To the external environment, this coupled model behaves almost like an atomic CHECKPOINT DEVS model, which represents an Essence Checkpoint which gets addressed by only one Activity(Space). The coupled model adds an additional output DEVS port. Two of the input DEVS ports of the coupled model are coupled to the MULTIACTIVITYCHECKPOINTALLOCATOR model (Table 3.8 on page 159) which routes – or *allocates* – the resources assigned to a given Activity(Space) to the respective CHECKPOINT model, which reflects the part of the whole MULTIACTIVITYCHECKPOINT which gets addressed by that specific Activity(Space). A MULTIACTIVITYCHECKPOINT is considered fulfilled by the simulation model only, if *all* of its partial CHECKPOINTS addressed by the different Activities are fulfilled. Partial CHECKPOINTS behave like described in 3.5.3. It is the purpose of the MULTIACTIVITYCHECKPOINTAGGREGATOR to aggregate the state of the partial CHECKPOINTS and to communicate that state to the other parts of the simulation. For that reason the output DEVS ports of this atomic model are coupled to the output DEVS ports of the coupled model.

CalendarTime

This atomic DEVS model serves as translator between simulation time, measured in ticks, and calendar time, to provide observers of the simulation and players of the game with a more natural perception of time elapsed in the simulation. This model has no input ports. The utilization of the 5 output ports is summarized in Table 3.10 on page 160. The DEVS model differentiates working days from weekend days. Start and end time of a working day are set as model constants. The model gets initialized with a calendar time representing the start of the simulation. If simulation runs are repeated, it is of particular importance, to start the simulation with same calendar time parameter, as results will vary otherwise.

Type	Name	Purpose	Quantity
IN	in_working_day_start	cf. Table 3.6 on page 155	1
IN	in_working_day_stop		1
IN	in_resources_assigned	cf. Table 3.6 on page 155, DEVS port gets coupled to all Activities addressing the Checkpoint, resources have to be allocated to respective partial Checkpoint	1
IN	in_effective_working_power_factor	cf. Table 3.6 on page 155	1
IN	in_preconditions_satisfied	cf. Table 3.6 on page 155	1
IN	in_preconditions_unsatisfied	cf. Table 3.6 on page 155	1
IN	in_alpha_state_preconditions_satisfied	cf. Table 3.6 on page 155	1
IN	in_alpha_state_preconditions_unsatisfied	cf. Table 3.6 on page 155	1
IN	in_lost	cf. Table 3.6 on page 155	1
OUT	out_checkpoints_preconditions_satisfied	cf. Table 3.6 on page 155	1
OUT	out_checkpoints_preconditions_lost	cf. Table 3.6 on page 155	1
OUT	out_fulfilled	inform containing AlphaState about the fulfillment of this aggregated Checkpoint	1
OUT	out_state_lost	cf. Table 3.6 on page 155	1
OUT	out_ui_message	cf. Table 3.6 on page 155	1
OUT	out_status	cf. Table 3.6 on page 155	1
OUT	out_completed_activity_<X>	inform addressing Activity(Space) about the fulfillment of this aggregated Checkpoint	dynamic: 1 per addressing Activity(Space)

Table 3.7: I/O Ports of MultiActivityCheckpoint DEVS Model

Type	Name	Purpose	Quantity
IN	in_resources_assigned	DEVS port gets coupled to all Activities addressing the Checkpoint, incoming resource assignments are allocated to respective partial Checkpoint addressed by the Activity(Space) that resources were assigned to (cf. Table 3.6 on page 155)	1
IN	in_lost	gets coupled to all Activities addressing the Checkpoint, insufficient resource assignments (cf.) may lead to lost checkpoints, lost message gets allocated to respective partial Checkpoint addressed by the Activity(Space) with insufficient resource assignment	1
OUT	out_resource_assigned_ <partial_checkpoint>	coupled to one specific partial checkpoint, routing of assigned resources to partial Checkpoint addressed by the Activity(Space), cf. Table 3.6 on page 155	dynamic: 1 per addressing Activity(Space)
OUT	out_lost_ <partial_checkpoint>	coupled to one specific partial checkpoint, routing of lost message to respective partial Checkpoint addressed by the Activity(Space) with insufficient resource assignment cf. Table 3.6 on page 155	dynamic: 1 per addressing Activity(Space)

Table 3.8: I/O Ports of MultiActivityCheckpointAllocator DEVS Model

Type	Name	Purpose	Quantity
IN	in_checkpoint_state	DEVS port gets coupled to all partial checkpoints, enables aggregation of partial checkpoints' states, (cf. Table 3.6 on page 155)	1
OUT	out_fulfilled	inform containing AlphaState and about the fulfillment of this Checkpoint, using state aggregated from partial Checkpoints, cf. Table 3.6 on page 155, Table 3.7 on page 158	1
OUT	out_lost	inform containing AlphaState and about the loss of this Checkpoint, cf. Table 3.6 on page 155, 3.7	1
OUT	out_ui_message	enable feedback and interactive use of the simulation model, using the aggregated state, cf. Table 3.6 on page 155	1
OUT	out_status	measurement and intelligent tutoring using aggregated state of partial checkpoints, cf. Table 3.6 on page 155	1

Table 3.9: I/O Ports of MultiActivityCheckpointAggregator DEVS Model

Type	Name	Purpose	Quantity
OUT	out_calendar_time	send current calendar time as DateTime object	1
OUT	out_start_working_day	signal the start of a new working day	1
OUT	out_end_working_day	signal the end of a working day	1
OUT	out_new_day_started	signal the start of a new calendar day (including weekend days)	1
OUT	out_status	measurement and intelligent tutoring, sending current simulation time as well as current calendar time inside the simulation	1

Table 3.10: I/O Ports of CalendarTime DEVS Model

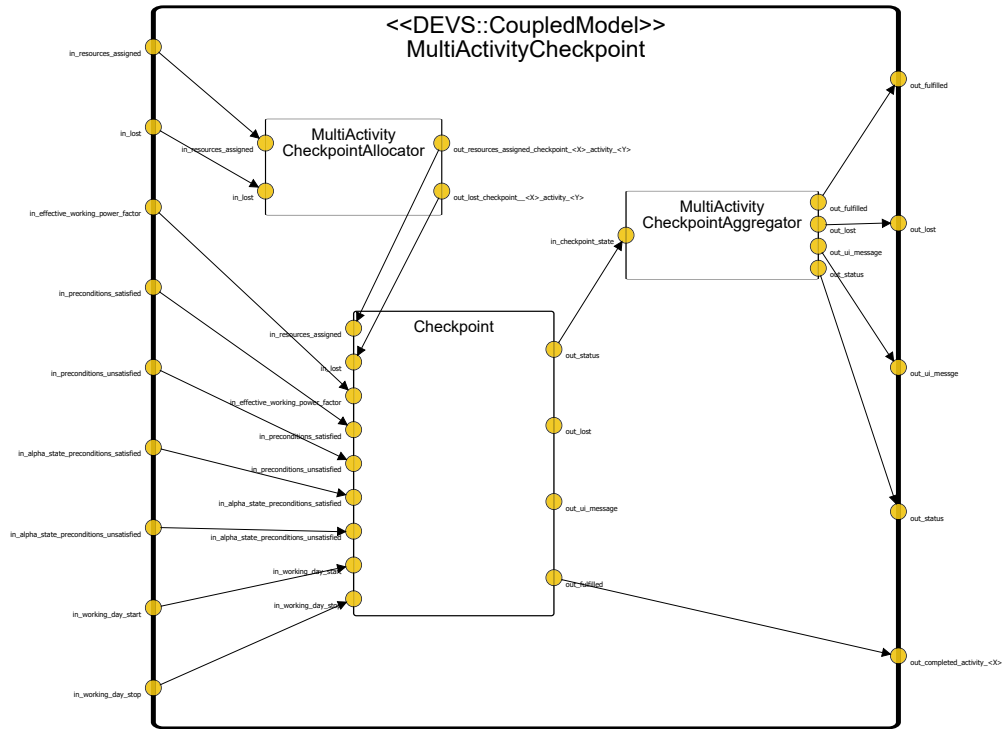


Figure 3.15: I/O Ports and Input/Output Coupling of MultipleActivity-Checkpoint DEVS Model

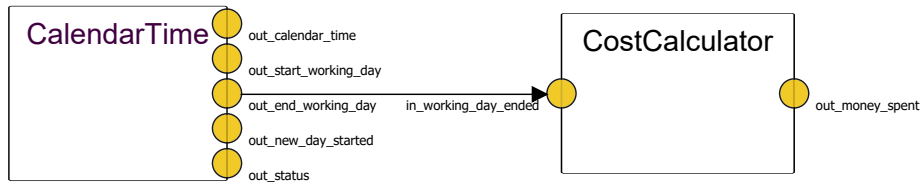


Figure 3.16: Connection of Atomic DEVS Models CalendarTime and Cost-Calculator

CostCalculator

The CostCalculator DEVS model already got described in 3.4.2. Figure 3.16 on page 161 illustrates the coupling of the CalendarTime model and the CostCalculator model enabling aggregating cost caused by progressing time.

AssignmentAllocator

The AssignmentAllocator DEVS model provides output ports for each of the ActivitySpace DEVS models. Model's input port *in_resource_assignment* is connected to the hierarchical simulation model's input port by an external input coupling (EIC). This coupling enables the interactive assignment of resources (virtual workforce) at simulation runtime to one or more ActivitySpaces. Such an assignment is only possible if the model is awaiting an assignment. It gets in this mode by receiving a message at input port *in_awaiting_assignment* through an exter-



Figure 3.17: I/O Ports of AssignmentAllocator DEVS Model

nal event. The internal couplings IC_{pause} (cf. 3.5.2) assure that such a message is received every time the simulation model gets paused.

Once the AssignmentAllocator receives a resource assignment represented by a set of variable pairs

$$\{(activity_i, resources_assigned_i) \mid i \in \mathbb{N}, 1 \leq i \leq 15\}, \text{ with}$$

$$\sum_{i=1}^{15} resources_assigned_i \leq MAX_PERSON_HOUR_AVAILABLE$$

the resources $resources_assigned_i$, represented in person-hours per working day, are assigned to each of the ActivitySpaces $activity_i$ via the output ports “out_activity_{*i*}”, each one coupled exclusively to its respective ActivitySpace.

SimPause

The SimPause DEVS model provides connected models of the simulation (cf. 3.5.3, 3.5.3) with the ability to stop the run of the simulation. This is necessary to enable an simulation observer, e.g. a player of the simulation game, to analyze the situation and to make decisions including resources assignments (cf. 3.5.3). The internal couplings IC_{pause} (cf. 3.5.2) assure that the simulation model gets paused at each message sent to the interactive user interface via the output port “out_ui_message” of the hierarchical simulation model.

Once the SimPause model receives a message at the “in_pause_simulation” input port, it sets its associated SimulationPauseSwitch’s PAUSE state to TRUE, signaling the SimulationController to pause this simulation. The process, controlling the simulator of the chosen simulation environment, was modified to observe and acknowledge the state of the SimulationPauseSwitch instance and to pause the simulation process when required. Once the SimPause model receives a message at the “in_resume” input port, the PAUSE state of its associated SimulationPauseSwitch instance gets set to FALSE signaling the SimulationController to resume the simulation until a next pause is requested. Figure 3.18b on page 163 illustrates the structure needed to accomplish this simulation process behavior. The UML class diagram shows the SimulationController, which controls the Simulator and holds an association to the SimulationPauseSwitch, which is also associated to the SimPause class representing an atomic DEVS model.

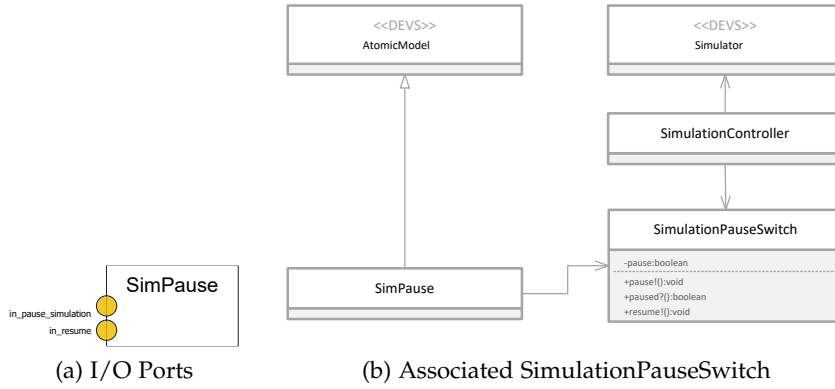


Figure 3.18: I/O Ports of SimPause DEVS Model

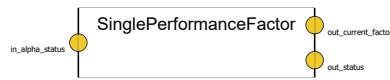


Figure 3.19: I/O Ports of SinglePerformanceFactor DEVS Model

SinglePerformanceFactor

The *SinglePerformanceFactor* DEVS model is utilized to represent one Alpha with its progressing states influencing the performance of the virtual workforce. The internal couplings IC_{perf} of the hierarchical simulation model (cf. 3.5.2) ensure that all relevant model elements are connected in a proper way. The model gets initialized with a set of state-factor parameters, value pairs each representing the factor, by which the performance of the team is influenced, if the respective state is reached. All *SinglePerformanceFactors* get aggregated in a *TotalPerformanceFactor* (cf. 3.5.3). A factor of 1.0 represents 100% and embodies a team performing at its full potential where a factor of 0.5 represents 50% of teams potential/nominal performance. Table 3.11 on page 164 shows an example configuration of *SinglePerformanceFactors* used in the case study evaluation of the *Simulation Game*. With this configuration the resulting total performance factor (cf. 3.5.3) at the begin of the simulation is $0.8 * 0.8 * 0.8 = 0.512$ representing a virtual workforce performing at 51.2% of its potential performance. Considerations for the quantification of the simulation model including performance factors are discussed in section 3.7.3. These factors are configurable to the needs of the respective context.

TotalPerformanceFactor

The *TotalPerformanceFactor* DEVS model is utilized to aggregate *SinglePerformanceFactors* (cf. 3.5.3). The actual performance of the virtual workforce is defined as

$$Performance_{actual} = Performance_{nominal} * TotalPerformanceFactor$$

Alpha	Alpha State	Factor
Team	Formed	0.8
	Collaborating	0.9
	Performing	1.0
Work	Prepared	0.8
	Started	0.9
	Under Control	1.0
Way of Working	In Use	0.8
	In Place	0.9
	Working Well	1.0

Table 3.11: Examples of SinglePerformanceFactors Used in Case Study Model

, where the nominal performance of the virtual workforce is set as parameter at model initialization and measured in person_hours/working_day. A virtual workforce with an a nominal performance of 24 person-hours/working day and a total performance factor of 0.5 (50%) has an actual performance of 12 person-hours/working day and is able to accomplish the results of Checkpoints that require an effort of 12 person-hours on one working day. Increasing workforce's performance by progressing the states of Alphas defined as SinglePerformanceFactors would eventually lead to an optimal total performance factor of 1.0 (100%) doubling the effort that may be done on a working day.

Figure 3.20 on page 166 shows the feedback cycles implemented in the simulation model. Accomplishing the results of a Checkpoint leads to its fulfillment. Once all Checkpoints of an AlphaState are fulfilled that AlphaState itself is reached. The current state of an Alpha gets transferred to its corresponding SinglePerformanceFactor where it gets evaluated and transferred to the TotalPerformanceFactor aggregating all SinglePerformanceFactors. The value of the current total performance factor resulting from the multiplication of all SinglePerformanceFactors gets fed back to all of the Checkpoints, increasing the performance of the virtual workforce at accomplishing the results required by the respective Checkpoint, and the cycle closes.

On the other hand a Checkpoint could get lost due to inactivity (cf. 3.3). A lost Checkpoint results in one or more lost Alpha States decreasing the corresponding SinglePerformanceFactor. The resulting decreased TotalPerformanceFactor gets again fed back to all of the Checkpoints, this time decreasing the performance of the vir-

tual workforce at accomplishing the results required by the respective Checkpoint.

Once an AlphaState changed, the potential resulting change in performance does not consume any simulation time. The potential new resulting total performance factor is effective for all Checkpoints without any delay in simulation time.

UIHttpSender

This DEVS model is responsible to transfer any user interface (UI) message and game day statistics, once per simulated calendar day, to an HTTP(S) endpoint. Used in the game environment, this model connects the simulation output with the game interface and game statistics. For that purpose, the model gets initialized with two URIs and an identification token. This UIHttpSender model has no internal transition function and depends solely on external events. Once a message at one of the input ports is received, that message gets decorated with some metadata, packaged, and delivered to the defined HTTP(S) endpoint.

SimulationReporter

This DEVS model collects data of simulation model's elements and stores them in a plain SimulationReport object that enables the IntelligentTutor model and the output of simulation day statistics via the UIHttpSender model. The internal couplings IC_{report} of the hierarchical simulation model (cf. 3.5.2) ensure that all relevant model elements are connected in a proper way to enable data collection.

IntelligentTutor

This DEVS model was designed to support a simulation user, e.g. a player of the simulation game, with guidance if needed. The model follows (only) basic concepts of *Intelligent Tutoring Systems* (cf. section 2.2.2 on page 23).

To enable such tutoring, the IntelligentTutor needs to know simulation's state and the underlying simulation model. Model's single input port is coupled to the CalendarTime model (cf. IC_{cal_time} in section 3.5.2) and receives a message at each end of a working day. The external transition function checks for need of advice and sends a tutoring message via its single output port as appropriate. To check for advice needed, the model makes use of the associated SimulationReport (cf. 3.5.3) and has access to a structure describing the simulation model.

Guidance is provided by the tutor after a configurable number of consecutive virtual days with unfavorable actions taken. The tutoring feedback is formulated as question or remark of the virtual workforce, not as an intervention from the outside, which would break the

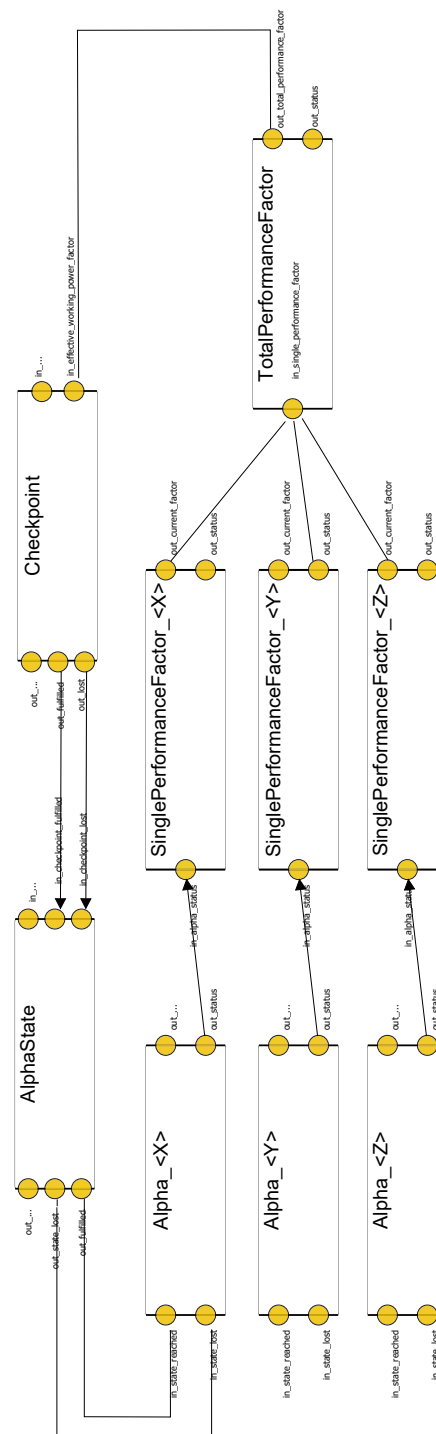


Figure 3.20: TotalPerformanceFactor DEVS Model

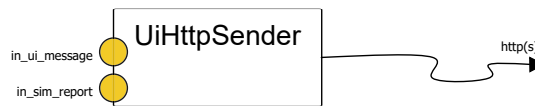


Figure 3.21: I/O Port and HTTPS Interface of UiHttpSender DEVS Model

Type	Purpose	Example
NO PROGRESS	inform the user of the model that no progress is made at all	<i>"It seems we did not make any progress for 7 days! Are you really sure we are focusing on the right activities?"</i>
ACTIVITIES WITHOUT PROGRESS	inform the user of the model that one or more of the activities with resources assigned do not accomplish any progress	<i>"We were assigned to 'Shape the System', 'Implement the System' and 'Test the System' but seem not to make any progress there. We are missing results of foregoing work. It seems we should do something different prior to this."</i>
INSUFFICIENT PERFORMANCE	inform the user of the model that the performance of the team could be increased (cf. 3.5.3, 3.5.3) by progressing Alphas of the Endeavor Area of Concern	<i>"We think we are not performing optimally and should care more about the Alphas Work, Team and Way of Working."</i>
ONLY MAINTENANCE ACTIVITIES	inform the user of the model that it is good to take care of Activities resulting in a good team performance but not sufficient to progress the SE endeavor	<i>"The activities 'Coordinate Activity', 'Support the Team', and 'Track Progress' get your team's performance high. Once this is achieved you can hold this performance with less effort per day and should care for other activities progressing other alphas too."</i>

Table 3.12: Tutor Message Examples

communication flow of simulation users, e.g. players of the simulation game, and the virtual workforce taking orders. It is intended to make the users of the interactive simulation think about actions taken, when they got stuck.

This approach takes the view that a tutoring message should just give hints to make the user aware of his missteps. To find strategies to master current challenges should be part of the problem solving and learning process supported by social interaction, e.g. with teammates while playing the simulation game (cf. 4.7.5, 4.7.6). It is not the goal of this approach, to provide an environment where a single learner gets guided solely by an artificial intelligence. Instead, the approach aims to provide an environment triggering social interactions and demanding cognitive effort. Getting stuck somewhere in the simulation/game may provide such a trigger and initiate a discussion of different attempts to find a solution in a given situation.

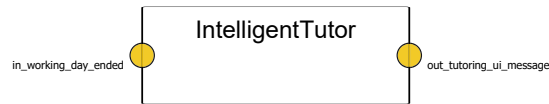


Figure 3.22: I/O Ports of IntelligentTutor DEVS Model

3.5.4 Complexity Considerations

It is always a good idea to have a clue of the dimensions of a simulation model. Taking the Essence kernel [188] to build a simulation model following the introduced modeling approach results in a hierarchical DEVS model with the following atomic/coupled DEVS models:

- 15 ActivitySpaces,
- 7 Alphas,
- 41 Alpha States,
- 207 Checkpoints including 24 MultiActivityCheckpoints (1 coupled model plus each including 4 atomic models),
- 3 SinglePerformanceFactors (cf. 3.5.3) and 1 TotalPerformanceFactor,
- 1 CalendarTime,
- 1 CostCalculator,
- 1 AssignmentAllocator,
- 1 SimPause,
- 1 UIHttpSender,
- 1 SimulationReporter and
- 1 IntelligentTutor DEVS model.

In sum such a model of the SEMAT Essence kernel contains 377 elements (353 atomic DEVS models and 24 coupled DEVS models in case of MultiActivityCheckpoints) and more than 3,500 coupling connections, including 59 defined interdependencies represented by preconditions (cf. 3.3) between them.

Obviously, these numbers would vary, if a modeler decided to define performance factors (cf. 3.5.3) and preconditions (cf. 3.3) in a different way. This amount of elements and connections between them reveal that a diagram containing all of simulation model's elements and their coupling relations would not be expedient at this place.

More important is the fact, that the number of concepts employed to build the simulation model is really manageable. The simulation

model utilizes the concepts of Essence. Some concepts inherent to Essence were made more explicit to be transparently represented in the simulation model. Only a few were added to complete the model and provide a foundation for the *Simulation Game* (cf. Section 4.7). Most of the complexity is hidden to the simulation modeler. Section 3.7 describes the modeling process in detail.

3.5.5 From Essence Kernel To Essence Method

The simulation approach was chosen to be as simple as possible to enable interactive exploration of Essence concepts. As a minimum foundation, the Essence kernel was chosen as a starting point. As the kernel includes all essential elements, resulting models are capable of conveying the basic Essence mechanisms, esp. the *PDCA cycle* (cf. section 2.16 on page 107) to steer an SE endeavor.

To enable the simulation approach to represent Essence methods too, further elements would have to be added to it. These include most notably

- *Work Products*, their *Levels of Detail*⁸, and their *Checkpoints*
- *Sub-Alphas*, driving or inhibiting kernel's 7 top level *Alphas*,
- concrete *Activities*, and
- *Competencies* with their *Competency Levels* required by *Activities*.

Work Products and their *Levels of Detail* could be handled conceptionally quite similar to *Alphas* and their *Alpha States* in the simulation model. They are the result of *Activities* performed resulting in fulfilled *Checkpoints* and eventually *Levels of Detail*.

Activities are concrete *ActivitySpaces*⁹ and would behave quite similar in the simulation model. Since a method would not necessarily be complete—with all *AlphaStates* being addressed by concrete *Activities*, potentially an additional *ActivitySpace* element addressing the remaining *AlphaStates* would be needed.

Sub-Alphas would behave in the simulation model as the existing *Alphas*, but a mechanism would have to be added since they do not necessarily exist at the start of an SE endeavor. Performing an *Activity* in the simulation model could create such a *sub-Alpha*, or just make it *visible* so that it can be progressed in further *Activities*.

Such extensions to the simulation model may be part of future work (cf. section 3.9 on page 177).

⁸ to some extent conceptionally similar to Alpha States of an Alpha

⁹ Actually Activity is a concrete AbstractActivity—the common base class of (concrete) Activity and ActivitySpace (the abstract placeholder for concrete Activities).[188]

3.5.6 Limits of Chosen Approach

To provide a simple and highly transparent simulation approach, some simplifications were made to the simulation model.

1. Currently, the parameters for setting the number of working hours per working day and the cost per working day can be set only once at simulation initialization resulting in a *fixed team size* throughout the simulation model. This would be limiting if a modeler would like to represent a varying team size at different phases of the SE endeavor.
2. One ActivitySpace currently allows only one active Checkpoint at each point in time. This could be limiting at the representation of simultaneous tasks. At the moment, two concurrent tasks can be represented by quantifying the second Checkpoint with an effort of 0 and omitting a STARTED-message of the second Checkpoint, which results in FULFILLED-messages of both Checkpoint at the same simulation time.
3. Currently the defined performance factors influencing virtual workforce's performance and hence the progress made in the virtual endeavor are influencing *all* activities (ActivitySpaces). This could be limiting if a modeler would like to restrict this influence to specific activities (ActivitySpaces) and omit it at others.
4. The simulation approach does currently not allow for an exit condition based on reached Alpha States. The duration and budget¹⁰ parameters are the only way to stop the simulation. This may be limiting in some circumstances and requires for a balanced quantification of the model.
5. The simulation approach does—by intention¹¹—not contain any stochastic effects and does not provide any formulas to be entered to define simulation behavior.
6. Currently the simulation approach supports only element types of the SEMAT Essence kernel, such as *Alphas*, *Alpha States*, *Checkpoints*, and *Activity Spaces*. An extension to support further elements to represent Essence methods may be part of future work (cf. section 3.5.5 on the previous page and section 3.9 on page 177).
7. This simulation approach does not release the modeler from thinking about interdependencies and messages provided as

¹⁰ defined at the game level

¹¹ This seemed not to be rewarding since the *Simulation Game* has to provide comparable results to enable competition between teams. These results should be solely dependent on players decisions.

feedback. Since descriptions of Checkpoints and feedback messages are based on free natural text the approach can't detect semantically unreasonable definitions.

8. Since the simulation approach follows a highly declarative approach based on Essence's elements, their (inter-)dependencies, and Checkpoints providing feedback messages, it may feel less accurate and fine-grained to modelers that are used to utilize more imperative approaches.
9. This simulation approach is—by intention—focused on the essentials as provided by the underlying SEMAT Essence standard. This may limiting to modelers who would like to represent specific incidents, e.g. two virtual team members leaving the team caused by a win on the lottery.

The scope of the simulation model was chosen consciously. None of these restrictions were perceived as too limiting at conducting the case study as all of them can be thematized at debriefing activities of the *Simulation Game*. Other objectives may require different characteristics of the simulation.

Since the Essence kernel represents all essential dimensions of an SE endeavor, including all important aspects in checklists of Alpha States, everything of essential importance should be considered. Since the kernel is extensible additional aspects, perceived by a modeler as essential too, could be added.

3.6 IMPLEMENTATION

The DEVS simulation was implemented by using the *Ruby* programming language¹² and utilizing the *DEVS-Ruby* modeling and simulation library¹³[232].

This library allows for an iterative development of simulation models using an API or a provided DSL and offers the opportunity to build an own DSL for the simulation of a specific domain on top of it. *DEVS-Ruby* is capable of providing basic visualizations of the simulation model built.

By not hiding the underlying programming language, it allows for a very flexible utilization and integration with complementing libraries and frameworks, e.g. testing or specifying frameworks¹⁴ to provide automated testing of simulation model's elements. That way it is enabled to be integrated with the intended game environment.

¹² <https://www.ruby-lang.org/en/>

¹³ <https://github.com/devs-ruby/devs>

¹⁴ The implementation utilizes RSpec [<http://rspec.info/>] to describe and test the intended behavior of the simulation model's elements.

3.7 STREAMLINING THE MODELING WORKFLOW

Existing approaches in simulation and DGBL in SE (process) education require remarkable training effort to enable the creation of simulation models—training efforts that are not of (direct) use outside of the specific simulation/game environment.

As already described in the guiding principles of this simulation approach, this one aims at providing synergies, simplicity and transparency, and efficient model construction and customization (cf. section 3.2 on page 117).

3.7.1 *Designing an Essence Kernel/Method*

The first step in creating the simulation model is to design the Essence kernel/method¹⁵ in a standard tool provided to the Essence community¹⁶.

In a second step, the output of this standard tool is parsed and stored as an Essence model at *meta-level 1 (M1)* (cf. section 2.7.2.1 on page 106). An enacted endeavor of this model (cf. section 4.6.3 on page 221 and figure 4.15 on page 224) serves as foundation of the simulation model.

3.7.2 *Making Interdependencies Explicit*

Essence defines dependencies and relationships of its elements through the provided language. Additionally, interdependencies between elements, esp. between Checkpoints and Alpha States represented by the semantics of Checkpoints' descriptions, have to be made explicit for the simulation model. This process already got described in depth in section 3.3 on page 120.

3.7.3 *Quantifying the Model*

To utilize the simulation model to drive the Simulation Game, several parameters have to be quantified. Table 3.13 on the facing page summarizes the parameters to be quantified by the modeler.

¹⁵ Currently the simulation model is capable only of Essence kernel elements, esp. *Alphas*, *Alpha States*, *Checkpoints*, and *Activity Spaces*. Future work may provide the additional elements to enable the simulation of Essence methods, including elements like *Activities*, *Sub-Alphas*, *WorkProducts*, and their *Levels of Detail*.

¹⁶ Currently supported is only the EssWork Practice Workbench of Ivar Jacobson International (IJI). "IJI is proud to support the SEMAT initiative and to offer its new practice authoring tool "The EssWork Practice Workbench" for use by the SEMAT community for FREE. Based on the SEMAT Method Architecture, EssWork Practice Workbench comes with an interactive version of the Essence Kernel, and enables the

for each instance of	Condition	Parameter	Meaning
Checkpoint	for each Activity Space addressing instance's Alpha State (cf. section 3.5.3 on page 144)	effort_to_do	effort in person-hours to accomplish the results represented by Checkpoint's description
Checkpoint	for each Activity Space addressing instance's Alpha State and representing an ongoing activity	effort_to_regain	effort in person-hours to (re-)accomplish the results represented by Checkpoint's description once it was lost
Activity Space	for all instances	checkpoints	sequence of Checkpoints that get fulfilled in that order if Activity Space gets performed with enough resources assigned (measured in person-hours per working day)
Activity Space	only for instances representing an ongoing activity (needing some kind of maintenance to hold a state once it was reached) (cf. section 3.5.3 on page 144)	checkpoints_to_lose	hash of consecutive inactive days each with a set of Checkpoints to lose if resources assigned to Activity Space are below min_ph_per_wd
Activity Space	only for instances representing an ongoing activity (needing some kind of maintenance to hold a state once it was reached) (cf. section 3.5.3 on page 144)	min_ph_per_wd	minimum person-hours per working day to be assessed as active (and maintaining a state once reached), no progress is made if assigned person-hours are below this parameter, state (Checkpoints) gets lost if assigned person-hours are below this parameter
Alpha State	only for instances that got identified to represent a performance factor (cf. section 3.5.3 on page 163)	factor	influences the performance of the virtual workforce if this instance is the most advanced Alpha State of an Alpha (cf. section 3.5.3 on page 163)
Simulation	once per simulation	duration	maximum duration of the simulation, measured in simulation ticks, simulation stops at reaching
Simulation	once per simulation	budget	maximum budget available, simulation stops at reaching
Simulation	once per simulation	ph_per_wd_available	person-hours of the virtual workforce available per working day that can be assigned to Activity Spaces
Simulation	once per simulation	cost_per_working_day	the cost of the virtual workforce per working day
Simulation	once per simulation	working_hours_per_working_day	the cost of the virtual workforce per working day

Table 3.13: Quantifying the Simulation Model—Necessary Model Input

At this moment, a modeler has to decide how much effort to put into the quantification to provide a realistic model. A systematic common lack of data that is generally interfering validation and verification of SPSM was already mentioned in section 2.8.1 on page 111. Dickmann et al.[71] state “a principal methodological problem: software-producing organizations are time-varying and can not be experimentally tested for different scenarios.”

With SEMAT Essence as a new approach, the chance to find empirical data aligned to it is even lower.

A modeler has different options at this point.

On the one hand, quantification data could be estimated based on a fixed scenario, e.g. by utilizing methods like COCOMO-2[31] or a variant of *Function Point Analysis*, and subsequently mapping the results to the Essence kernel, namely the Checkpoints of the Alpha States of the seven kernel Alphas. This would be a rather expensive exercise and the benefit of such an undertaking might be questionable. Needless high efforts may discourage educators to use such simulation approaches.

As Navarro[174] states with regards to educational simulation models, “At the expense of some realism, effects need to be somewhat obvious and ‘over the top’ at times in order to effectively illustrate and enforce the concepts being taught.”

A more pragmatic approach producing a less “accurate” but reasonably quantified model may provide, not the same but a “good enough” effect for the intended purpose. For the intended purpose—to get familiar with Essence concepts and to utilize the Essence kernel to assess the current state of an SE endeavor and to steer it towards a set target state—basically follow the *PDCA cycle* (cf. section 2.7.2.1 on page 105), it does not matter if a given Checkpoint would need one person hour or ten. Such effort to accomplish the result represented by the description of a Checkpoint should rather be seen as variable since it will vary from one unique endeavor to the next.

From such a perspective a deep (artificial accurate) quantification of the simulated model for educational goals seems neither affordably feasible nor advisable. Rather should simplifications of the simulation model be thematized in debriefing actions of the *Simulation Game*. With this standpoint the taken approach follows recommendations of Zhang et al.[313] who propose to set expectations and prediction accuracy as well as precision in parameter values into relation with the context of the modeling purpose and desired results (cf. section 2.8.1 on page 111).

Nonetheless, to provide a holistic perspective, the quantification of the model should ensure that it is beneficial to care for the Alphas of

creation of kernel extensions and practices as well as their composition into methods.”[247]

the *endeavor* Area-of-Concern (*Work, Team, Way of Working*). By setting them as an input of *SinglePerformanceFactors* (cf. section 3.5.3 on page 163) they influence the performance of the virtual workforce hence affect the progress made by the virtual team. Following this approach results of activities are accomplished less efficient until the virtual team gets performing well (cf. section 3.5.3 on page 163).

The model has to be quantified and balanced in a way, that development activities last long enough to compensate the effort done to accomplish results of *endeavor*-Alphas' Checkpoints.

This effort and the needed "maintenance effort" of ongoing activities to retain *endeavor*-Alphas' Alpha States have to be lower than the gains in efficiency.

Checkpoints addressed (via their Alpha States) by the Activity Space *Operate the System* might get quantified with a very high "effort" since they result in a *Retired Software System* that might seem not adequate in this context. Alternatively, the *budget* provided (cf. table 3.13 on page 173) may be balanced in a way that the simulation stops at a desired (maximum) state.

3.7.4 Modeling Feedback for Usage in Game Environment

To enable a player in the *Simulation Game* to interact with the game and hence the underlying simulation, feedback about current events has to be provided. To provide this feedback it has to be modeled before.

Table 3.5 on page 154 summarizes the types of messages that can be defined by the modeler for each Checkpoint. To enable the player to interact and to reason about the progress made to assess Checkpoints and Alpha States, at least the *FULFILLED*-message has to be provided. Should the Checkpoint be addressed by an ActivitySpace via its AlphaState, the at least additional *LOST*- and *REGAINED*-messages are required. The more messages for the several types are provided, the more feedback is presented to the player of the *Simulation Game*.

At defining the messages, a modeler defines the level of difficulty to map it to its Checkpoint. Depending on the learning objective, such a message can be formulated rather closely to the description of the Checkpoint or rather different from it and hence making it more challenging to map it to its Checkpoint.

The model used in the case study described in chapter 5 on page 251 rather made use of messages formulated close to the Checkpoint descriptions making it rather easy to map messages to Checkpoints and their state of progress.

3.7.5 *Game Scenario Narrative*

If an elaborated game scenario narrative gets utilized, the messages providing feedback (cf. section before) may incorporate aspects of it and hence provide a more integrated impression.

While such incorporated narrative feedback may result in an even more immersive game experience, it requires remarkably more effort at modeling and lessens the chance to reuse it in other contexts.

From a learning theory perspective, such incorporated narrative feedback may provide more “situation” to a *Situated Learning* approach (cf. section 2.2.3 on page 24). At the same time, it might result in more *extraneous cognitive load* (cf. section 2.2.5.1 on page 31) interfering a focus on the essential learning content.

3.8 DIFFERENCES TO EXISTING MODELING APPROACHES

Compared with existing modeling approaches utilized in DGBL in SE education, this approach is—with SEMAT Essence—the first one to utilize a standard to unify SE process/method description and communication.

This approach does neither provide a DSL nor a rule editor as existing approaches. A modeler in this approach does not have to build a simulation from scratch by creating formulas, rules, and other environment-specific modeling elements.

Modeling in the introduced approach is much more like designing a screenplay with a timeline of events and progress made in an SE endeavor, represented by Essence Checkpoints and messages of the virtual team—actually multiple timelines since the defined interdependencies of the Checkpoints and Alpha States allow for multiple ways through an SE endeavor.

Having to quantify each checkpoint and to provide a number of messages to each checkpoint requires some effort. But different from existing approaches, a modeler—familiar with concepts of the Essence kernel—will need much less training effort compared to existing approaches since this approach is by intention simple and transparent. A lecturer should not have problems at explaining the behavior of the underlying simulation model represented in the *Simulation Game* to any player. The approach added as less as possible new concepts on top of Essence concepts. That way, thinking about the simulation model should be to the largest extent be thinking about Essence kernel concepts themselves.

3.9 FUTURE WORK

Currently, the simulation approach supports only element types of the SEMAT Essence kernel, such as Alphas, Alpha States, Checkpoints, and Activity Spaces.

To support the simulation of Essence methods, this approach has to be extended to provide more element types of the Essence language, e.g. sub-Alphas, WorkProducts, their Levels of Detail, and concrete Activities.

Such an extension to the simulation approach may be integrated into future work, if feedback collected at future deployments of the approach reveals such demands.

The second field of future work is related to the modeling process of the approach. While it is yet streamlined at the conceptional level, it is not yet fully supported technically resulting in manual mapping effort needed. Such effort is planned to be removed from the process to enable a wider public use of the approach. Development work has already been started but has to be extended and integrated into the environment of the approach.

AN INTEGRATED APPROACH TO LEARN SOFTWARE ENGINEERING METHODS

This chapter introduces the *Integrated Approach to Learn Software Engineering Methods*, hereafter short *Integrated Approach*, developed in this research work. Before it gets described in depth—with its concepts, assumptions, phases, developed digital learning games, and supporting tools—its motivation gets explained in detail. Since the approach is based on SEMAT Essence, Essences' contribution to learning gets examined with regards to learning theories and competencies demanded by curriculum guidelines. The chapter closes with a comparison of approaches taken and related work.

4.1 THE CASE FOR AN INTEGRATED APPROACH

By comparing Jonassen's summarized characteristics of constructivist learning environments (cf. section 2.2.3 on page 28) with the characteristics of (well designed) course or capstone projects, it gets obvious that both may provide a large intersection.

But as described in chapter 1 on page 1, reality shows that there are gaps to close. This *Integrated Approach* was designed to support at closing them.

As stated in section 1.3 on page 7 following observations can be made in course projects:

- It is not easy for students to orientate inside of a given SE method or software process.
- It is quite hard for students to answer the question(s) who, how, when and in particular *why* specific activities should be accomplished.
- A rigid fixation on technological and functional details lets students lose the holistic view on the development process as a whole—with all its relevant dimensions.
- All too frequent, takeaways from such projects are too specific and hardly transferable to other contexts and upcoming challenges.

It can be observed that many students are leaving course projects with a fuzzy impression that applying SE methods/practices/tools could

have been helping to reach even better results if they would have been employed in a more rigorous, disciplined and goal-oriented way.

Losing a holistic view due to remarkable cognitive load and just delivering artifacts requested by a pre-chosen software process under time pressure, hardly provides the deep impression that practices, methods, and tools were truly supporting students' work.

Preceding research showed that simulation and games were able to foster empathy with the necessity of employing software processes to accomplish more complex software projects to a certain level successfully. But evaluation of those approaches showed too that they, in general, were not able to impart new knowledge.[301]

None of the approaches known to the author of this thesis utilized the opportunities provided by simulation and DGBL to prepare students for their real project work in a concrete and direct way.

Figure 2.14 on page 105 gives a hint that less experienced learners, or team members in a software project, with yet fewer capabilities, need more explicit guidance by SE practices or software processes. That is why a "*just let them figure out, how to accomplish the tasks*" approach combined with limited time may not be an option for students, who are not already experts in their field. As we know from cognitivist learning theory, minimally guided or even unguided instruction may not provide ideal learning experiences (cf. section 2.2.3 on page 29).

But just providing a software process description to support students may not be the ideal solution for several reasons too:

1. Process descriptions usually provide sequenced activities aligned to phases and milestones. They provide descriptions of *who* (which role) should do *what* and *how* to accomplish tasks. But they usually lack a common structured and goal-oriented underlying concept to provide the *why* of these activities and to assess the whole SE endeavor in a holistic way.
2. Just getting familiar with *one* process description does not provide the demanded transferable knowledge being of use in all future SE endeavors.
3. To some extent software process descriptions are similar to process worksheets used to guide learners through complex tasks. Van Merriënboer and Sweller[165] explain "Because the information provided in a process worksheet typically has high element interactivity, simultaneously performing the learning tasks and consulting the worksheet may be too demanding. Working memory demands may be increased further because learners must split their attention between the task and the process

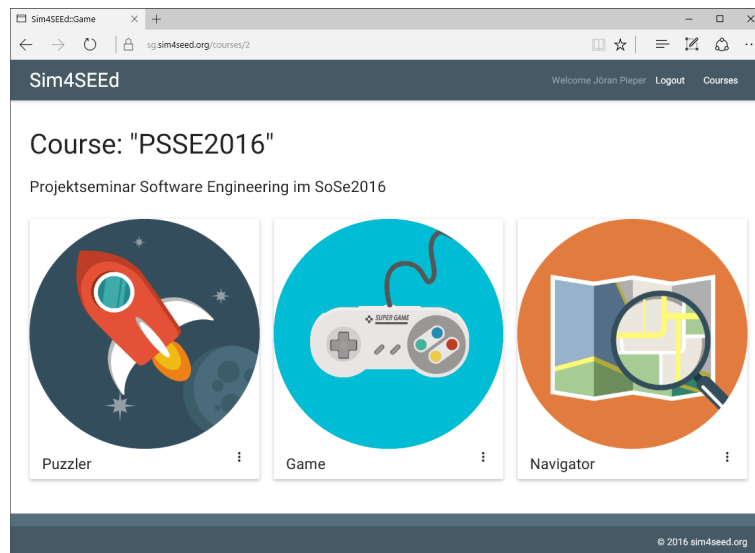


Figure 4.1: Screenshot of the Course Environment: Tools Supporting Phases 1 – 3

worksheet.” They propose that learners “thoroughly study the recommended phases and hints before they start to work on the learning tasks as suggested by the results of studies [...]”[165] That way “a cognitive schema may be constructed in long-term memory that can subsequently be activated in working memory during task performance. Retrieving the already constructed schema should be less cognitively demanding than activating the externally presented complex information in working memory during task performance.”[165]

Adding the SEMAT Essence kernel on top of a software process description or providing it instead may significantly reduce problems (1.) and (2.) mentioned above. But this approach would either add even more cognitive load on top and boost problem (3.) mentioned above or take away the guidance of practices needed by less experienced learners if the kernel would completely replace defined practices or a software process.

The Integrated Approach introduced in this chapter aims at addressing the problems mentioned above by applying learning theory¹, utilizing SEMAT Essence, simulation, and DGBL. The following sections describe foundations, characteristics, phases, and elements of this approach.

Figure 4.1 shows the environment provided to course members containing the tools to support the first three phases of the *Integrated Approach* that get described in following sections.

¹ with a focus on constructivist approaches but without neglecting insights of cognitive learning theories arguing against minimally guided instruction

4.1.1 *Facilitating NOT Replacing Existing (Project-Based) Learning Approaches*

It is important to emphasize that this *Integrated Approach* utilizing DGBL aims not at replacing existing project-based learning approaches like course or capstone projects. Instead, it has been explicitly designed to

1. facilitate them by *lowering the cognitive load* students are faced with at conducting their project work,
2. provide more *opportunities for reflection and social interaction* to facilitate learning,
3. *increase the transferability* of knowledge gained by utilizing the generalist SEMAT Essence kernel approach considering all essential dimensions of SE endeavors in a practice and process independent way, and
4. let students *experience the orientation and guidance* provided by the chosen approach to enable the development of *SE attitudes*, where the provided practices and methods are not just perceived as further cognitive load but as truly supporting at keeping a holistic perspective on the SE endeavor.

4.1.2 *Learning Theories And Educational Approaches Applied*

The *Integrated Approach* is based on SEMAT Essence. Essence's characteristics providing support to learning and contributing to learning objectives of curriculum guidelines are described in section 4.2.2 on page 191.

This section provides a description of learning theories and educational approaches applied in the *Integrated Approach* beyond that.

The *Integrated Approach* has been designed to provide a constructivist learning environment. It provides Jonassen's[134] characteristics of constructivist learning environments that are facilitating learning (cf. section 2.2.3 on page 24). Table 4.1 on the next page summarizes the representation of those characteristics by elements of the *Integrated Approach*. Those elements are described in detail in following sections. Different from most existing approaches (cf. section 2.5 on page 68) the *Simulation Game* sets a focus on social interaction (cf. section 2.2.3 on page 24), adding concepts of social constructivism to the approach.

The *Phase 2* of the *Integrated Approach* connects *Phase 1*, where motivation, basic concepts, and relationships of the Essence kernel get introduced with the *Phase 3* where students apply it actively in their practical project work. Thereby the *Simulation Game* used in *Phase 2*

Jonassen's [134] characteristics of constructivist learning environments ^a facilitating learning	Representation in the Integrated Approach
provide multiple representations of reality to avoid oversimplification	learning content is provided at different abstraction levels and from different perspectives throughout the <i>Integrated Approach</i> ,
represent the complexity of the real world	by utilizing the whole Essence kernel the complexity and all essential dimensions of an SE endeavor are provided right from the start
emphasize knowledge construction not knowledge reproduction	while Phase 1 is addressing primarily memorization of factual and conceptual knowledge the following Phases 2 and 3 are primarily addressing procedural and metacognitive levels requiring to understand and apply learned concepts, learners are provided with interactive environments allowing a high degree of freedom at scaffolded discovering and decisioning how to proceed
emphasize authentic tasks in a meaningful context rather than abstract instruction out of context	context and authentic tasks are provided through case study (<i>Phase 1</i>), <i>Puzzler</i> (<i>Phase 1</i>), interactive participation in <i>Simulation Game</i> including debriefing (<i>Phase 2</i>), and real project work (<i>Phase 3</i>)
provide learning environments such as real-world settings or case-based learning instead of predetermined sequences of instruction	<i>Simulation Game</i> provides a (virtual) real world setting, powered by the Essence kernel and hence representing all relevant essential dimensions of an SE endeavor, real project work provides such an environment per se (within the limitations of academic settings)
encourage thoughtful reflection on experience	reflection is encouraged throughout the <i>Simulation Game</i> by providing collaboration and competition features, immediate feedback and a <i>reflective game type</i> ^b , regular team meetings while running the course project utilize the checklists provided by the Essence kernel to guide discussion and to assess the SE endeavor encouraging holistic reflection on achieved results in all relevant dimensions of the SE endeavor
enable context- and content-dependent knowledge construction	by providing an <i>inherent</i> ^c <i>Simulation Game</i> where learning content is integral part of the gameplay, by utilizing the gained knowledge in following project work situated in a given context
support collaborative construction of knowledge through social negotiation	facilitated through collaborative features in the <i>Simulation Game</i> (cf. sections 4.7.2.3 and 4.7.5), debriefing activities following the <i>Simulation Game</i> , and teams' discussions guided by the Essence kernel in real project work (<i>Phase 3</i>)

Table 4.1: *Integrated Approach* Addressing Jonassen's[134] Characteristics of Constructivist Learning Environments Facilitating Learning

^a cf. section 2.2.3 on page 24

^b cf. section 4.7.11 on page 245 for game characterization

^c cf. section 4.7.11 on page 245 for game characterization

serves as an important *anchor* in the sense of *Anchored Instruction* (cf. section 2.7.2.1 on page 105).

Following the *Elaboration Theory* (cf. section 2.2.5.2 on page 34) learning content should be introduced from *general to detailed*, from *simple to complex*, and from *abstract to concrete*.

The *Integrated Approach* presents the whole Essence kernel, its motivation, basic concepts and their relationships in *Phase 1*. At this stage, it keeps rather abstract and general.

Phase 2, utilizing the *Simulation Game*, provides the opportunity to deepen and apply the concepts in a virtual project. By practicing the *PDCA cycle* (cf. section 2.7.2.1 on page 105) to drive their virtual SE endeavor, students have to observe kernel's *concrete* elements in more *detail*. By utilizing their relationships, things get more *complex*, e.g. by having to consider inter-relationships and multiple paths that can be taken to progress the Alphas of their endeavor.

Students in *Phase 3* apply the Essence kernel or method practically in their project work. While the feedback from the virtual team was relatively easy to map to checkpoints of Alpha States, students now have to assess the state of their real SE endeavor and to think more deeply about the semantics of Checkpoints, Alpha States and Alphas. Additionally, they have to choose and/or follow SE practices to progress their endeavor and thereby to accomplish real SE tasks. At this stage, the learning content presents itself in its most *concrete*, *detailed*, and *complex* form.

A primary objective of the *Integrated Approach* is to lower the *cognitive load* students are faced with in their real course or capstone project work to enable experiences where SE tools and methods are perceived as really supportive providing orientation as well as guidance to drive a software project.

By providing already familiar tools, concepts, and procedures students are faced with a lower cognitive load right from the start of their project work enabling them to experience orientation and guidance and to be more focused on particular challenges of their project domains.

At designing the *Integrated Approach* all phases of the learning process, as defined by Gagné's *events of instruction* (cf. section 2.2.2 on page 22), were considered. Table 4.2 on the next page summarizes how elements of the approach are addressing Gagné's events of instruction.

Event of Instruction	Addressing Element(s) Of the <i>Integrated Approach</i>
1. Gain attention	introducing interactive lecture and case study to introduce and motivate, <i>Essence Kernel Puzzler</i> (Phase 1)
2. Inform of objectives	
3. Stimulate recall	at introduction to <i>Simulation Game</i> and course project work
4. Present stimulus	<i>Essence Kernel Puzzler</i> (Phase 1), <i>Simulation Game</i> (Phase 2), real project work (Phase 3)
5. Provide learner guidance	through short concepts descriptions introducing levels of the <i>Essence Kernel Puzzler</i> (Phase 1), through integrated supporting tool (<i>Essence Navigator</i>) in <i>Simulation Game</i> (Phase 2), through already familiar concepts, procedures and tools right at the start of course project work (Phase 3)
6. Elicit performance	by providing challenging tasks in <i>Simulation Game</i> (Phase 2) and course project (Phase 3)
7. Provide feedback	direct feedback provided in multiple ways throughout the <i>Simulation Game</i> fostering social interaction inside the team, integrated into the course project
8. Assess performance	throughout the <i>Essence Kernel Puzzler</i> (Phase 1), the <i>Simulation Game</i> (Phase 2), as part of course project work (Phase 3)
9. Accommodate retention and transfer	through repeated application in <i>Simulation Game</i> (Phase 2) and real project work (Phase 3), through debriefing activities after the <i>Simulation Game</i> , by utilizing the SEMAT <i>Essence</i> kernel integrating process- and practice-independent concepts

Table 4.2: Elements Of the Integrated Approach Addressing Gagné's Events Of Instruction

4.1.3 Learning Objectives and Learner Profiles

4.1.3.1 Learning Objectives

The objective of this *Integrated Approach* is not just to provide knowledge. This approach aims at providing *competencies*, which were defined in section 2.3 on page 38 as:

A cluster of related knowledge, skills, and attitudes enabling a person to accomplish all the tasks in a given context, that correlates with measurable performance, which can be improved by education, training and experience.

Knowledge in this definition represents, *what one knows*. Skills in this definition represent, *what one can do*. This includes *capabilities* and *abilities* and involves *knowledge*. Attitudes in this definition are concerned with the ability to apply knowledge and skills in an effective manner, characterized by attributes of behavior such as *willingness, initiative, communicativeness, cooperativeness, self-reflection, trustworthiness, empathy, cultural and social sensitivity, professionalism, problem awareness, creativity* (if needed), etc. (cf. section 2.3 on page 38).

This clarification is important. Students should not just *know* something *about* SE methods, or the Essence kernel. They should be able to *apply* concepts (*to do*) and furthermore develop an *attitude* that lets them apply their skills and knowledge.

As described in section 4.2 on page 189 the proposed *Integrated Approach* is based on SEMAT Essence². How SEMAT Essence, esp. the Essence kernel, contributes to learning objectives of curriculum guidelines' knowledge areas and how it supports additional guidelines gets discussed in detail in section 4.2 on page 189.

While the *Phases 1 to 3* are aiming at providing *factual, conceptual, procedural, and metacognitive levels* of the *knowledge dimension* and the *remember, understand, and apply levels* of the *cognitive process dimension* of the *Revised Bloom's Taxonomy* (cf. section 2.3.1 on page 38), further phases may address higher cognitive levels. The approach introduced in this thesis is focused on the first three phases, which should be part of any undergraduate SE curriculum. Higher learning levels are rather part of graduate or highly specialized undergraduate curricula and may get addressed by *Phases 3+X* building on learning experiences made in *Phases 1, 2, and 3*.

After the first three phases of the *Integrated Approach* students should be able to basically utilize the Essence kernel, esp. apply the *PDCA cycle* (cf. section 2.16 on page 107), practically in an SE endeavor. They should know, when and how to apply it and furthermore have devel-

² In its initial version focused on the SEMAT Essence kernel. Integrating SE practices and methods based on SEMAT Essence may be the subject of future work.

oped an attitude to *appreciate* to utilize the orientation and guidance delivered by its characteristics.

To enable such competency, students have to master the terminology and understand the concepts provided by the kernel. They have to *understand* (inter-)relationships between its elements that provide kernel's characteristics. To *apply* the kernel in a course project they need (basic) skills to utilize it to holistically assess and steer an SE endeavor. On the *metacognitive* knowledge level students should be able to reflect their learning progress, identify knowledge gaps and their preferred resources to close them as well as to choose and use those procedures and tools that contribute the most to their teams' way of working.

It would not be realistic to expect students after performing learning activities of the *Integrated Approach* to act like SE professionals with years of experience. It would not be realistic too, to expect that every student remembers all the Alpha States, their Checkpoints and Activity Spaces addressing them. But as described in section 2.7 on page 95 and section 4.2 on page 189 this is not necessary to start utilizing the Essence kernel in practice since it provides practical and tangible scaffolding to practitioners.

Table 4.3 on the following page summarizes the primary learning objectives of the single *Phases 1 to 3* of the approach. While *Phase 1* is primarily addressing the *factual* and *conceptual* levels of the *knowledge dimension* as well as to some extent the *metacognitive* level of the *remember* level, *Phases 2 and 3* are primarily addressing the *procedural* and *metacognitive* level of the first two levels of the cognitive process dimension and all knowledge levels of the *apply* level of the cognitive process dimension. At the same time they are contributing to the deepening of knowledge of the other levels, since students are using elements and concepts over and over again.

The remaining levels of the cognitive process dimensions may get addressed by additional *Phases 4 to X*.

With regards to the *Software Engineering Competency Model (SWE-COM)*[14] (cf. section 2.3.2 on page 44) this *Integrated Approach* may contribute to the acquisition of the required skills of the crosscutting skill area "*Software Process and Life Cycle Skills*"³ as well as to the acquisition of *cognitive skills (reasoning, analytical skills, problem solving, innovation)* and to *behavioral attributes and skills*, both with regards to SE methods.

³ Directly if it gets interpreted in a future-oriented way anticipating an orientation towards flexible composable SE practices instead of monolithic software processes. Indirectly in both cases by providing the SEMAT Essence kernel as thinking framework supporting the acquisition of new software processes.

knowledge	metacognitive	Phase 1 Phase 2 Phase 3	Phase 2 Phase 3	Phase 2 Phase 3	Phase 3+x	Phase 3+x	Phase 3+x
	procedural	Phase 2 Phase 3	Phase 2 Phase 3	Phase 2 Phase 3	Phase 3+x	Phase 3+x	Phase 3+x
	conceptual	Phase 1 Phase 2 Phase 3	Phase 1 Phase 2 Phase 3	Phase 2 Phase 3	Phase 3+x	Phase 3+x	Phase 3+x
	factual	Phase 1 Phase 2 Phase 3	Phase 1 Phase 2 Phase 3	Phase 2 Phase 3	Phase 3+x	Phase 3+x	Phase 3+x
		remember	understand	apply	analyze	evaluate	create
cognitive process							

Table 4.3: Learning Objectives of Phases of the Integrated Approach Mapped to the Revised Bloom's Taxonomy[13, 147]

With regards to the *Software Engineering Body of Skills (SWEBOS)* (cf. section 2.3.2 on page 45), proposed by Sedelmaier and Landes[243] and focusing on *context-sensitive soft-skills* required for SE, this *Integrated Approach* especially contributes to the *competencies for structuring one's own way of working* and to one of the three identified top soft-skills: *the comprehension of the complexity of software engineering processes and understanding of cause-effect relationships*.

4.1.3.2 Learner's Profiles

It is assumed that students starting with *Phase 1* are already familiar with developing software in smaller contexts. Ideally they should be already familiar with at least some of the practices required in an SE endeavor and are about to start a course or capstone project. The *Integrated Approach* introduced in this chapter was explicitly designed to support their learning.

Although the Essence kernel does not explicitly require deeper SE knowledge before starting to become acquainted with, it may remain too abstract otherwise. The Essence kernel defines six *Competencies*. Each of the *Competencies* already known by the students, offers some kind of recognition and may ease their steps.

But even without already owning those competencies, students may start at an earlier level as all essential elements get defined and described in the kernel.

Since today's students likely were born after the 1980s, they likely share some characteristics of the *digital natives* or *net generation* (cf. section 2.4.2.4 on page 63) and "prefer inductive reasoning, want frequent and quick interactions with content, and have exceptional

visual literacy skills—characteristics that are all matched well with DGBL.”[295]

4.2 BASED ON SEMAT ESSENCE

SEMAT Essence was already introduced in section 2.7 on page 95. To provide a thorough foundation for the *Integrated Approach*, SEMAT Essence’s support to learning should be indicated.

The following section examines the claim of SEMAT proponents that Essence facilitates SE education.

To find support or disapproval for these claims, concepts and principles of learning theory as well as curriculum guidelines, introduced in section 2.2 on page 19, are consulted.

4.2.1 *Experiences With Utilizing Essence in SE Education*

Ng and Huang[184] reported that they conducted a workshop with professors and PhD students at Chinese universities. Within that workshop, they performed some exercises connected to SEMAT Essence with the participants. They state that

- “Participants immediately saw the importance of having a consensus on terminology,”
- “Essence is able to encompass a wide range of challenges, particularly the typical ones in software engineering. This implies that Essence provides a sufficient breadth of coverage of software engineering disciplines.”
- “There was a Eureka moment, when participants saw that they could represent different lifecycles by shifting the alpha states around.” and
- “Participants recognized immediately that the business-social aspects of software engineering represent a huge void in their curricula. More importantly, they saw Essence as a way to describe the scope of a curriculum, and the scope of each course.”

Ng and Huang[184] conclude that “Universities cannot teach everything that industry requires. But university can provide students with a firm grasp of the fundamentals and give them the tools to learn and understand the diversity of software engineering later in their career.” This statement fits well with the needs of 21st-century learning skills described in section 2.2.4.

Jacobson et al.[118] report of utilizations of the SEMAT Essence kernel in first- and second-year SE courses at KTH Royal Institute of

Technology in Sweden. In a first-year course, students went through the Alphas to evaluate the results of their conducted projects. In second-year courses, students used the kernel to assess their running projects. The realizing lecturers conclude that the kernel assures the consideration of all essential aspects of an SE endeavor and that the students were able to “easily identify the good and bad sides of their development methods.” They summarize “Because they had to follow all the kernel alphas, the students could learn the total scope of the software engineering endeavor and thereby know what will be required of them in their professional careers.”[118]

Unfortunately, the authors provide no further details of the studies.

Pénaire and Sedano[207] provide the first detailed report of a field study utilizing the Essence kernel in multiple student projects. Participating students worked in co-located or distributed teams and already had a considerable average work experience ranging from 3 to 10 years. The SE process was chosen by the teams themselves based on students’ “reasonable knowledge of a diverse set of generally accepted software engineering practices, and the ability to execute these practices somehow effectively.”[207] All teams chose an iterative process model for their projects and were supported by faculty facilitators at Essence meetings.

Pénaire and Sedano describe challenges participants faced at using the Essence kernel. They mention ambiguities in checkpoint descriptions perceived by students, e.g. students asked “What do they mean by ‘enough’?” or “What kind of ‘constraints’ are they talking about?”[207]. They report that these ambiguities were leading to situations “where the team discusses the meaning of a checklist item instead of having a conversation about the project.” Another challenge identified was the combination of iterative software development approaches and the process- and practice-independent Essence kernel, which defines Alphas and their states at a project or release level but—without adding concrete Essence practices—not at the level of single iterations.

The authors[207] report that 90% of responding students appreciated the approach and 80% said they will use it in future projects. The faculty in charge of the course projects attested “a much better early project organization with lot less floundering.” Pénaire and Sedano[207] concluded that the Essence kernel approach “provides student teams with a simple, lightweight, non-prescriptive and method-agnostic way to examine their projects holistically, structure team reflections, manage risks, monitor progress and steer their projects. [...] most effective during project initiation and for monitoring and steering the work done at the project or release level.”

4.2.2 *Essence's Support of Learning Theories And Educational Approaches*

The Essence kernel approach should provide characteristics facilitating learning theories. This section examines its adequacy from the perspective of learning and educational theories.

4.2.2.1 *Elaboration Theory*

Reigeluth's *Elaboration Theory* (cf. section 2.2.5.2) holds that instruction, or learning content, should be organized from *general to detailed*, from *simple to complex*, and from *abstract to concrete*.

With the *Separation of Concerns* principle applied (cf. section 2.7.2 on page 97), the Essence approach provides a natural fit for utilizing the *Elaboration Theory* and supports all three sequencing requirements.

The kernel providing only the common, *general*, and essential elements of every SE endeavor packaged into *simple* concepts is separated from *complex*, varying *details* of varying practices. Hence Essence inherently facilitates the organization of learning content from *general to detailed* and from *simple to complex*.

To provide a method- and practice-independent kernel the authors had to define its elements in a general and *abstract* manner providing the opportunity to define *concrete* practices and eventually methods based on its foundation. For instance, that is why the kernel does not provide *concrete* activities, defining the *how* of tasks to be done, but Activity Spaces, the *abstract* placeholders for them defining the *what* and *when*⁴ of tasks to be done.

4.2.2.2 *Anchored Instruction*

Anchored Instruction (cf. section 2.2.3 on page 27) provides support to assimilating new knowledge by providing a common *anchor*, which new learning content can be related to.

In a wider sense—by utilizing the same principles—the Essence kernel can provide such an *anchor* throughout a whole SE career for students introduced to it. Each new SE practice, SE method, or SE process, students are faced with in their career, can be mapped to the Essence kernel, which facilitates the identification of already known and new elements as well as the synthesis of those elements.

4.2.2.3 *Cognitive Load Theory*

Sweller's *Cognitive Load Theory* (CLT) holds that the working memory of learners is very limited in capacity as well as duration and can hold only a few items at the same time including even less new knowledge items. CLT holds too that information from, virtually unlimited, long-term memory vastly extends capability of working memory (cf.

4 provided by addressing *Alpha States*, which are well-ordered

section 2.2.5.1 on page 31). As Paas and Sweller[195] state “the extent to which working memory limitations matter depends on the extent to which the information being dealt with has been organized in long-term memory.”

From this perspective the support of learning provided by the Essence kernel is twofold. By separating the common essentials from (practice dependent) details fewer concepts and elements can be learned faster to get productive. Once familiar with them they provide the ability to compare new practices or methods to already existing structured knowledge (cf. *schema theory* in section 2.2.2 on page 21). These knowledge structures in long-term memory support the working memory in the cognitive architecture of the learner to recognize patterns, similarities as well as differences in varying SE practices and hence to categorize and learn new practices and methods more efficiently.

4.2.2.4 *Constructivist Learning*

Since the Essence kernel was designed to be *tangible*⁵, *practical* and *actionable* (cf. section 2.7.2.1 on page 104) it provides the opportunity to *learn by doing*. An Essence user does not have to learn all elements, e.g. Alphas, their states and Checkpoints, in advance to use it. Different from SE processes just documented somehow and somewhere, a team dynamically uses the *actionable* kernel, e.g. by progressing states of all relevant dimensions (Alphas).

With these characteristics, the Essence approach provides support for *constructivist educational approaches* (cf. section 2.2.3 on page 24). Starting to use the Essence kernel in a software project provides kinds of *Active Learning*, *Learning by Doing* and *Discovery Learning* as a team incrementally digs deeper into kernel’s elements and discovers more and more of their relationships on demand and as needed in that situation. Since projects naturally provide authentic activities resembling or representing real-life environments, it provides characteristics of deeply *Situated Learning* too.

One criticism with *Situated Learning*, actually too contextualized learning, is that it would not “promote flexible transfer. The transfer literature suggests that the most effective transfer may come from a balance of specific examples and general principles, not from either one alone.”⁶[58] Utilizing the Essence kernel in a software project provides both specific examples in the context of the project domain *and* general principles combined in the kernel hence the approach provides both support for driving the respective project as well as transferable knowledge of use in future endeavors.

⁵ e.g. by cards (Alpha card, Alpha State card, etc.) provided for its elements

⁶ cf. section 2.2.3

4.2.3 *Essence's Support of Curriculum Guidelines*

Curriculum guidelines are designed by experts in the field concerned with SE education. To examine this question three curriculum guidelines are consulted:

1. the IEEE/ACM guideline for undergraduates SE2014,
2. the German guideline for undergraduates and graduates published by the *German Informatics Society (GI)* GI2016, and
3. the guideline for SE graduate degrees GSwE2009 (cf. section 2.3 on page 38).

None of the guidelines refers to SEMAT Essence in one of the knowledge areas to be covered in curricula. This is not surprising since the specification is newer than two of the guideline recommendations.

In addition to the knowledge areas to be covered by curricula, the guidelines provide a set of specific *curriculum guidelines*⁷ representing demanded characteristics of a curriculum. If Essence should fit into the curriculum guidelines, it should contribute to the accomplishment of the demanded characteristics of a curriculum.

4.2.3.1 *SE2014*

SEMAT Essence would contribute to the knowledge area “*PRO—Software Process*” of the SEEK (cf. section 2.3.3 on page 46).

With a set of twenty curriculum guidelines describing demanded characteristics of curricula, the SE2014 provides the most guidance for curriculum construction. Those related to the topic of SE process or methods are discussed in this section:

“Curriculum Guideline 5: Learning certain software engineering topics requires maturity, so these topics should be taught toward the end of the curriculum, while other material should be taught earlier to facilitate gaining that maturity.”

Using the *SEMAT Essence Kernel* as an *anchor*, in the sense of *Anchored Instruction* as described above, enables a wide range of activities that can be related to the kernel as an *anchor*. Such an approach provides a coherent and holistic view on SE endeavors. Looking at the kernel from multiple perspectives and deepening the knowledge through discussions about different aspects, contributes to a constructivist learning environment. The usage of the Essence kernel is not limited to one single course. Courses addressing different knowledge areas of SE may share the *Essence Kernel Anchor*.

⁷ Unfortunately, the term *curriculum guideline* is confusingly used both as the title for the whole guideline documents and specification of desired characteristics and qualities of a curriculum following the approaches in the respective document.

“Curriculum Guideline 10: Software engineering problem solving should be taught as having multiple dimensions. An important goal of most software projects is meeting client needs, both explicitly and implicitly. [...] Problem solving is best learned through practice and taught through examples.”

The SEMAT Essence kernel provides Alphas to represent the essential elements, or dimensions, of each SE endeavor in a very compact way. Its approach is holistically considering all dimensions, since all Alphas, by definition, have to be progressed in a balanced way. With the provided Alphas *Stakeholders* and *Opportunity* the alignment towards *meeting clients needs* is deeply integrated into the Essence kernel (cf. section 2.7 on page 95).

An essential design goal of Essence has been to support software practitioners. The Essence kernel is actionable and well suited to provide guidance in any course or capstone project, independent from the method or practices chosen by the lecturer or students. Case studies that are already available provide examples to discuss and learn from.

“Curriculum Guideline 11: The underlying and enduring principles of software engineering should be emphasized, rather than the details of the latest or specific tools. [...] In a good curriculum, it is the enduring knowledge in the SEEK topics that must be emphasized, not the details of the tools. The topics are supposed to remain valid for many years; as much as possible, the knowledge and experience derived from their learning should still be applicable 10 or 20 years later. Particular tools, on the other hand, will rapidly change. It is a mistake, for example, to focus excessively on how to use [...] the detailed steps of a methodology[...]. Applying this guideline to processes (also known as methods or methodologies) is similar to applying it to languages. Rather than memorizing the details of a particular process model, students should be helped to understand the goals being sought and the problems being addressed so they can appropriately evaluate, choose, and adapt processes to support their future work as software engineering professionals. [...]”

The SEMAT Essence kernel does exactly, what is demanded by this guideline. By applying the design principle of *separation of concerns* (cf. section 2.7.2 on page 97), it separates a *stable kernel*, providing common ground “independent of the kind of software endeavor, the complexity of their requirements or software system, and the size of their team”[118], from practices, describing a specific way how to tackle single aspects following today’s, yesterday’s, or even tomorrow’s state of the art. This means, getting acquainted with the kernel once, provides a thinking framework of long-lasting validity and value. While the practices representing the current state of the art may change over time or are likely to change over time, there is always a common ground facilitating to acquire new approaches.

By utilizing Essence, *steps of a process model* get reflected by a modified state of the endeavor, represented by progressing the Alpha States. This way each activity, or step of a process model, is oriented towards the goal to progress Alphas of an endeavor—fostering a structured and goal-oriented attitude.

Based on common ground, it is much easier for learners to realize common goals and different approaches to reach them in varying process models, which simplifies their evaluation, selection, and adoption.

“Curriculum Guideline 12: The curriculum must be taught so that students gain experience using appropriate and up-to-date tools, even though tool details are not the focus of the learning. Performing software engineering efficiently and effectively requires choosing and using the most appropriate computer hardware, software tools, technologies, and processes (collectively referred to here as tools). Students must develop skill in choosing and using tools so they go into the workforce with a habit of working with tools and an understanding that selecting and developing facility with tools is a normal part of professional work. Appropriateness of tools must be carefully considered. Tool selection should consider complexity, reliability, expense, learning curve, functionality, and benefit. Tool selection also needs to consider educational value and usefulness in the workplace after graduation. [...]”

To choose from processes or methods, a student needs to be aware on the implications of using or not using their inherent practices. Common ground, independent from any practice, method, or process, that provides a holistic view supports students and practitioners to get aware of aspects addressed by a method—and those that are omitted. Students (as well as any practitioner) get an impression, if and how particular Alpha States get addressed in the chosen method and what the resulting implications of a candidate method are.

“Curriculum Guideline 13: Material taught in a software engineering program should, where possible, be grounded in (a) sound empirical research and mathematical or scientific theory or (b) widely accepted good practice.”

The SEMAT Essence Kernel is the result of an effort to find a kernel of widely agreed-on elements. The list of signatories, individual experts in the field, companies representing industry, and universities around the world, proves wide agreement and input from various points of view.

“Curriculum Guideline 14: The curriculum should have a significant real-world basis. Incorporating real-world elements into the curriculum is necessary to enable effective learning of software engineering skills and concepts. A program should incorporate at least some of the following:

- *Case studies: Exposure to real systems and project case studies is important, with students taught to critique these examples and to reuse the best parts.*
- *Project-based activities: Some learning activities should be set up to mimic typical projects in industry. These should include group work, presentations, formal reviews, quality assurance, and so forth. It can be beneficial to include real-world stakeholders or interdisciplinary teams. Students should understand and be able to experience the various roles typically found in a contemporary software engineering team.*
- *Capstone project: Students should complete a significant project [...] in order to practice the knowledge and skills they have learned. Building on skills developed in other project-based learning activities, students should be given the primary responsibility to manage this capstone project [...]. Team projects are most common and considered to be best practice because students can develop team skills that have value in many professional environments.[...]"*

SEMAT Essence has already been utilized by companies in real-world project settings. Case studies of these adoptions as well as educational case studies[244] are provided by SEMAT. As already mentioned, the SEMAT Essence kernel was designed to support practitioners in software development. The kernel is compact and lightweight and supports *Discovery Learning* and as such well suited to support project-based activities as well as capstone projects in a curriculum.

"Curriculum Guideline 16: Software process should be central to the curriculum organization and to students' understanding of software engineering practice." This guideline subsumes a set of demanded characteristics that should get examined in detail:

- *"Evolution of software process best practice: It is important to note that this curriculum guideline does not endorse any particular software process. Software process has evolved over the years, and it is reasonable to expect that this will continue. Assuming that one particular process or style of process is the best or final answer seems akin to making a similar assumption about a particular programming language or operating system. Every curriculum, while covering software process in depth, should also give students an appreciation of the range of processes and the notion that best practice in this area has and will continue to change."*

The difficulty to select one, or a small set, out of the wide range of processes/methods available today was already discussed (cf. section 2.6 on page 83). To give students an appreciation of the range of processes a common ground, like the one provided by the Essence

kernel, seems necessary to provide the chance to *understand* differences between methods' approaches. The chance to get lost in the wide range of methods, all using different formats and varying semantics connected to the vocab is very high otherwise. At recognizing the continual change in this area, students should be provided with a thinking framework with characteristics uncoupled from the speed of change and supporting adoption of change as required by a particular context.

- *"Range of software processes: Addressing the range of software processes implies that the curriculum give students some understanding of a selection of processes that might include, for example, both plan-based and agile methods. [...] The selection of processes to cover should reflect current industry practice."*
- *"Software process in context: The curriculum should address the relationship of software process and other elements of a work environment. For example, the supportive (or limiting) role of software development tools in successful processes use should be addressed as part of the coverage of tools. Students also need to learn about the importance of organizational culture, team and product size, and application domain in process selection. Environmental considerations such as these provide the context needed for students to understand the range of processes. Examples might include the continued presence of plan-based processes in the development of large embedded hardware/software systems or the advantages of agile processes in domains where requirements are incompletely understood or rapidly changing."*

To *understand* the range of processes, their differences on the different dimensions of an SE endeavor, represented by Alphas in the Essence approach, have to be understood. By supporting the comparison of Alpha States addressed in various phases or at different milestones of various processes, students, as well as any SE practitioner, are provided with a valuable aid. While differing processes can be described using the same dimensions (Alphas)⁸, they address different Alpha States at different phases in the endeavor.

By mapping the concrete Activities provided by the specific process to ActivitySpaces and addressed Alpha States, students, as well as any SE practitioner, are enabled to quickly grasp how complete the chosen method is with respect to all the Alpha States that have to be addressed. Such an exercise may reveal the blind spots of a method under observation that may have to be closed by complementary practices.

- *"Motivating software process: Like many aspects of software engineering, process is difficult to motivate until students understand central challenges such as scale, complexity, and human communication that*

⁸ at least at a high level

motivate all of software engineering. This has two implications for designing the curriculum. First, process needs to be introduced gradually. Making software process part of student work early in the curriculum helps to develop good habits. But process use must be carried through to later courses when student appreciation for process has been developed. Second, student work must [...] develop an appreciation of the challenges that motivate software engineering.[...] This might include not only student team project work but also case studies and observation of working systems."

The SEMAT Essence kernel provides a holistic picture of SE endeavors. Recognizing the associations between Alphas (cf. figure 2.12 on page 102) students get a first impression of the highly interconnected character of SE endeavor dimensions right from the start. Using the kernel as anchor to introduce partial aspects of SE, e.g. human communication aspects at requirements elicitation or team dynamics, keeps a holistic view and avoids to provide just single pieces of a puzzle. As already mentioned, SEMAT provides a number of case studies.

- *"Depth and application of software process: An appreciation of the range of processes should be combined with the development of the skills and knowledge to apply at least one particular process. Programs may need to focus on one particular process for students to achieve any proficiency by graduation. Even basic proficiency will only develop through repeated exposure across the curriculum, including student use of process in their own project work, team-based projects, and projects of sufficient scale to make process use meaningful."*

As this guideline states, learning needs time and repetition. Time is a very restricted resource in any curriculum. Instead of just introducing one process (out of so many existing) the transferability of knowledge and skills acquired is highly increased with adding the Essence kernel to the learning. Getting introduced to its compact structure, reduced to the essentials, more likely provides the skill to recognize similar patterns or activities addressing same objectives in other methods than with getting to know just one process with unique vocab and semantics.

- *"Process improvement: Addressing the range and context of software process provides a foundation for students to learn that software processes are not static, but rather they are something to be selected, managed, and improved. The curriculum should build on this foundation and directly address concepts of process improvement. Doing so requires students to understand process as an entity and an example of abstraction. Making that leap opens software process to concepts of modeling, analysis, measurement, and design that are central to process improvement."*

SEMAT Essence is provided to “put teams in control of their methods”[118]. By including the *Alpha Way of Working* the method, seen as a kernel plus a set of practices, itself is continuously assessed for their health and progress. Teams are encouraged to drop a practice that appears not to contribute to a well working way and add another, one more appropriate to the given context.

“Curriculum Guideline 18: Software engineering education needs to move beyond the lecture format and to consider a variety of teaching and learning approaches.”

“Curriculum Guideline 19: Important efficiencies and synergies can be achieved by designing curricula so that several types of knowledge are learned at the same time. [...] process, quality, and tools: Students can be instructed to follow certain processes, [...] as they are working on exercises or projects when the explicit objective is to learn other concepts. In these circumstances, it would be desirable for students to have had some prior introduction to relevant processes, tools,[...] so that they know why they are being asked to include them. The learning could be reinforced by following the exercise or project with a discussion of the usefulness of applying the particular technique or tool. The depth of learning of the process is likely to be considerable, with relatively little time being taken away from the other material being taught.”

The material provided by SEMAT supports more than just lectures. Cases studies are provided to discuss and learn from. Working with the Essence Kernel enables a style of *Discovery Learning* as described above. By using the introduced kernel as *anchor*, many activities in a course or curriculum can be related to it to provide a coherent, holistic view. The kernel supports project activities and capstone projects letting students experience the support given by utilizing this structured, goal-driven, and highly transferable approach. Since the kernel is not bound to any practice or method, it can be used in any curriculum environment, regardless of whether the focus taken is favoring more traditional plan-driven or agile practices.

Besides those curriculum guidelines, the *SE2014* authors demand a capstone project: *“The Capstone Project A capstone student project is regarded as being an essential element of a software engineering degree program. Such a project provides students with the opportunity to undertake a significant software engineering task, deepening their understanding of many of the knowledge areas forming the SEEK, and with a significant experience at the “a” (application) level of the Bloom taxonomy of learning.”*

At conducting the capstone project, the Essence kernel can provide guidance and facilitate a disciplined, goal-driven and structured approach, regardless of the method chosen. By utilizing the kernel, or a tailored Essence method, students are supported to keep a holistic

view on the endeavor while working concentrated and focused on specific SE tasks producing varying artifacts.

4.2.3.2 *GI2016*

The *GI2016* (cf. section 2.3 on page 38) is much more focused on cognitive competencies in provided knowledge areas and does not provide detailed curriculum guidelines, in the sense of the ones provided by the *SE2014*. The *GI2016* mentions the *SE2014* as source contributing to the guideline.

SEMAT Essence would contribute to the software process related competencies included in the knowledge area “*Software Engineering*” of this curriculum guideline.

The *GI2016* demands: “not solely to facilitate imminent topics but topics underpinned by theory enduring current trends and enabling lifelong learning.”[95] ⁹ This requirement largely corresponds with *Curriculum Guideline 11* of the *SE2014*, so the remarks given there apply here as well.

GI2016 demands the competency to transfer knowledge acquired to different contexts. Concerning SE methods this requirement to some extent corresponds to the *Curriculum Guidelines 11 and 12* of the *SE2014*, so the remarks given there apply here as well.

In the description of the knowledge area *software engineering* the *GI2016* demands that students should be able to distinguish between plan-driven and iterative approaches and should be able to assess the suitability of an SE method according to a given context. This requirement largely corresponds to the *Curriculum Guideline 16* of the *SE2014*, so the remarks given there apply here as well.

As the *SE2014*, the *GI2016* demand for project activities, hence the remarks given at the *SE2014* apply here as well.

4.2.3.3 *GSwE2009*

SEMAT Essence would contribute to the knowledge areas “*I - Software Engineering Management*” and “*J - Software Engineering Process*” of this curriculum guideline.

The *GSwE2009* does not provide detailed curriculum guidelines, in the sense of the ones provided by the *SE2014*. But concerning the distinction to the *SE2014*, the guideline states: “Every topic in *GSwE2009* must be mastered at level 2¹⁰ or higher. Moreover, many more topics in *GSwE2009* require mastery at level 3 than does *SE2004*; e.g., in *SE2004*, the topic of software process is addressed only at levels 1 and 2. In *GSwE2009*, the same topic is covered at levels 2 and 3.”[269]

⁹ Translated from German: “Ferner werden nicht nur gegenwartsnahe Inhalte vermittelt, sondern theoretisch untermauerte Konzepte und Methoden, die über aktuelle Trends hinweg Bestand haben und zum lebenslangen Lernen befähigen.”[95]

¹⁰ representing Bloom’s Taxonomy levels

With this statement and in the absence of comparable guidelines, it can be assumed that the general guidelines of *SE2014* with regard to SE processes apply to the *GSwE2009* too, with an even stronger focus on skills and the *Application* Bloom Taxonomy level. Essence addresses knowledge units in knowledge areas *I-Software Engineering Management* as well as *J-Software Engineering Process* (cf. table 2.9 on page 52) both primarily demanded to be mastered at the *Application* level.

The learning objectives *LO-8* and *LO-9* (cf. table 2.10 on page 53) of the *GSwE2009*[269] demand a student finishing a curriculum to “be able to learn new models, techniques, and technologies as they emerge, and appreciate the necessity of such continuing professional development” and to “be able to analyze a current significant software technology, articulate its strengths and weaknesses, compare it to alternative technologies, and specify and promote improvements or extensions to that technology.” Concerning SE methods, these requirements largely correspond to *Curriculum Guidelines 11,12, and 16* of the *SE2014*, so the remarks given there apply here as well.

The *GSwE2009* demand for a capstone experience, like *SE2014*, so the remarks given there apply here as well.

4.2.4 Conclusions

SEMAT Essence and its kernel provide support at learning considering a number of constructivist educational approaches as well as different educational theories. It contributes directly to a high number of requirements and guidelines of the presented curriculum guidelines and supports at reaching objectives of others.

This indicates that SEMAT Essence, esp. the Essence kernel, provides a solid foundation for the *Integrated Approach* introduced in this chapter.

Phase	learning objectives	supported by
<i>Phase 1:</i> getting familiar	learn the (common) vocab, know objectives and advantages of the Essence approach, name elements and assign correct definitions, know associations between elements of the Essence Kernel	case studies, literature, interactive short lectures supported by live quizzes, <i>Essence</i> <i>Kernel Puzzler</i>
<i>Phase 2:</i> practice Essence virtually	advance and deepen knowledge, orientate inside given method/kernel, apply concepts, control progress and health of endeavors alphas	<i>Simulation Game</i> , <i>Essence Navigator</i>
<i>Phase 3:</i> practice Essence in real project	apply concepts in real project, actually execute SE tasks, social aspects and dynamics of teamwork, control progress and health of alphas	<i>Essence Navigator</i>
<i>Phase 4..x:</i> from consumer to producer	compare practices and methods compose own methods out of existing practices, create own practices and kernel extensions, ...	standard Essence tools, e.g. <i>EssWork Practice</i> <i>Workbench</i> [115]

Table 4.4: 3+X Phases of the Integrated Approach

4.3 THE PHASES OF THE INTEGRATED APPROACH

The *Integrated Approach* presented in this thesis is structured into 3+X phases. These single phases build on each other. Phases 1 to 3 serve as an introduction into Essence and the Essence kernel and let students apply the concepts practically in virtual and real project environments. Additional *Phases 4 to X* provide further deepening of knowledge and let students outgrow of their role as pure method consumers. Table 4.4 summarizes the phases, their learning objectives, and tools chosen and designed to support the learning process.

The following sections describe the single phases of the approach in more detail.

4.3.1 Phase 1: Getting Familiar

In this phase students start to get familiar with Essence and the Essence kernel. Objectives, advantages and the fundamental structure of the kernel are introduced.

Available literature, case studies, as well as an introductory lecture, are used to engage student's interest. Multiple sources for literature

including the well readable specification[188] and multiple case studies showing the practical application of the Essence Kernel in different project phases are available [118, 244].

To provide an interactive introduction to the Essence kernel particularly Alphas and their interrelationships the *Essence Kernel Puzzler* (cf. section 4.5 on page 207) was developed in this research project. This game runs without installation in current web browsers. After successfully playing the game students should be able to name elements of the Essence kernel, to assign definitions correctly and to establish connections between kernel's elements.

The cost-free available web-based *Kahoot!*¹¹ is utilized to support highly interactive, motivating and engaging lectures in this phase. *Kahoot!* is a “game-based blended learning and classroom response system” integrating game-based and social approaches without the need for any special infrastructure since students use their own smart devices to interact. The resulting spirit of competition using an on-line quiz arouses emotions and gets very engaging in author's experience. Students get immediate feedback about their individual state of knowledge. Lecturers get immediately informed about how well different concepts already got understood in the whole course group. Wrong answered questions provide reasons for discussion to resolve misconceptions.

Once this phase is finished, the fundamental concepts and elements of the Essence kernel should be known to all participants. Depending on the focus of the course, single concepts, e.g. an Alpha of particular interest, may already have been discussed in more detail.

4.3.2 Phase 2: Practicing Virtually

In this next phase, the learned concepts get applied to illustrate their functional interplay.

The gained knowledge gets deepened and advanced in this phase. An interactive and collaborative digital learning game based on the simulation model, introduced in chapter 3 on page 117, gets used.

It is the objective of the *Simulation Game* to manage a simulated software project which uses the Essence kernel. The players have to explore the kernel—like they would in an adventure game—to employ the discovered and collected method elements, esp. Activities¹², to progress the virtual software project.

In doing so, it is important to do the right things (Activity Spaces) in proper sequence, to think about all relevant dimensions (Alphas) and their interdependencies, and to assess the current state of an endeavor consciously.

¹¹ <https://getkahoot.com/how-it-works>

¹² Activity Spaces respectively

Players control the virtual project through assigning activities to the virtual project team and actively monitoring and analyzing their feedback.

Feedback of the non-player characters (NPC) enables the assessment of Alpha States in the game. According to NPCs' feedback, Alpha States can be assessed, and Alphas progressed toward the desired end state. Players are supported by checklists of the respective Alpha States provided by the *Essence Navigator* utilizing the Essence kernel.

To explore the Essence kernel, players are using the *Essence Navigator*. This tool enables the exploration of the kernel and provides short descriptions of kernel's elements.

Furthermore, it enables the player to assess the Alpha States of the project via provided checklists. The *Essence Navigator* was designed with the objective to be used in the virtual game environment as well as in a real project environment. So it provides a linking element between this phase and the next *Phase 3* of the proposed *Integrated Approach* helping students to build upon already familiar concepts and procedures.

The usage of a standard, like Essence, and a standard tool is clearly favored over other approaches since training effort invested at this place is of value not only inside a specific game environment.

Adding the idea of teams to the individual gameplay of every participant opens new possibilities for interaction. In such a setting each team member explores the entire game and integrated learning content on her own.

Measuring the performance of the whole team as primary success indicator of the game fosters collaboration and discussion inside the teams. Providing dashboards of team scores, as well as an in-team ranking of individual players, gives orientation to players how optimized chosen paths are in relation to other one's results. This orientation gives immediate feedback and is the starting point for further interactions inside a team.

At this point the simulation game offers a quality that a real project could not provide in this compact way—learners get feedback much earlier and can learn from failures. By combining individual play and team play a learner can furthermore learn from experiences of teammates. As such the game provides a foundation for a „probe, hypothesize, reprobe, rethink cycle“[93], which leads up to a deeper involvement with the simulated SE method/kernel.

After finishing this phase successfully, all students have applied the acquired concepts in a defined simulated context. Inevitably they had to orientate inside a given SE method/kernel. They had to consider all essential dimensions (Alphas) of the SE endeavor and assessed their progress and health in a structured and continual way.

Alphas and their states gave guidance and recommendations for respective next steps. The game environment fostered collaboration and contributed to discussions and ideally reflection about different approaches in the game. The results of the game were analyzed and discussed course-wide in a debriefing activity. After that, students are well prepared for the next phase in the proposed *Integrated Approach*.

4.3.3 Phase 3: Practicing in Real Projects

It is the objective of this phase to apply Essence in a real project.

Students experience a remarkable cognitive load in course and capstone projects. Although the scope of such projects is usually limited according to the academic setting, such projects are challenging in many ways. Such cognitive load can be overwhelming. Students are at the risk of resorting to a deadline-driven way of working, fixed only on delivering demanded artifacts without any reflection on activities to be done.

Such a way of working limits the ability to transfer acquired experiences to other contexts. With the Essence kernel, which is already familiar to the students, project teams have a tool and thinking framework that provides orientation. The kernel supports to ask the right questions and to pay attention to all relevant dimensions of the endeavor.

The *Essence Navigator*, already known from the *Simulation Game*, provides a familiar environment and orientation. *Assessment Poker* may facilitate a conscious way of working of the whole project team. Using this technique, all team members assess the current state of the Alphas of the endeavor individually. The comparison of the individual assessments quickly reveals disagreements about the overall state of the Alphas and is a valuable reason for discussion and exchange of viewpoints. The homogeneity of individual assessments may be an indication for more or less successful communication inside the project team and may be another reason for discussion.

If regular assessments of projects' progress done by lecturers are part of the course concept, additional adjustment and feedback may be given.

After finishing this third phase of the *Integrated Approach*, students applied Essence concepts in a real project environment. Different from the application in the virtual environment in the second phase, students actually executed SE practices and produced real SE artifacts as demanded by the chosen SE practices. Social aspects and dynamics of teamwork were experienced. The described arrangements facilitated a conscious way of working where all team members regularly thought about the overall state of the endeavor. The progress and health of all essential dimensions regarding Alphas and their states

were assessed by the team in a structured and continuous way. At this point, it can be assumed that principally all students are capable of applying an Essence method/kernel practically in a project environment.

4.3.4 *Phases 3+x: From Method Consumer to Method Producer*

Depending on the focus of the curriculum, additional learning activities may be arranged. One obvious activity is to practice another SE method based on Essence virtually or practically and to compare and discuss the results with those from *Phase 2* and *Phase 3*.

Different approaches of SE methods and software processes would become visible and be an interesting object of comparison, analysis, and discussion.

Letting students create their own SE methods supported by tools, would enable them to switch from a pure consumer role into a producer role.

Composing available Essence practices into Essence methods or creating own practices from scratch deepens the knowledge and goes beyond the application of given SE practices and methods.

Insights gained at the second and third phase of the proposed approach would be of use. Different points of view could be thematized in following discussions.

Such learning objectives, esp. to produce own methods or practices, are beyond the scope of typical undergraduate curricula but may be interesting to graduate or rather highly specialized undergraduate curricula.

4.4 INTEGRATION INTO A CURRICULUM

Course or capstone projects are demanded by all curriculum guidelines (cf. section 2.3.3 on page 46). The *Integrated Approach*, esp. with its *Phases 1 and 2*, was primarily designed to introduce students to SE methods, with the support of the Essence kernel, and prepare them for their course or capstone project work. As such it naturally fits into a curriculum when SE methods get introduced and/or students are about to start their course/capstone project work. Since the approach was designed not to consume time excessively, it should be acceptable to integrate its *Phases 1 and 2* directly in front or as part of the course/capstone project, which would then represent *Phase 3*.

This integration is expected to provide the most benefit and was field-tested in the case study described in chapter 5 on page 251. Its description may serve as a blueprint of further deployments.

Learning objectives of *Phases 4 and higher*, esp. to produce own methods or practices, are beyond the scope of typical undergradu-

ate curricula but may be interesting to graduate or highly specialized undergraduate curricula. In such a case *Phases 4 and higher* were expected to follow *Phase 3*—or *Phase 2* if a course project should not be part of the deployment of the *Integrated Approach*.

Following a *spiral curriculum* approach as suggested by Bruner[47], the first and second phase of the *Integrated Approach* may support students to learn basic concepts of the Essence kernel first early in the curriculum. They could get introduced to more and more details iteratively as they walk through their curriculum and their learning is progressing. In that case, the Essence kernel could provide an *anchor* motivating to acquire those competencies as well as respective context for their practical application in an SE endeavor. As Ng and Huang[184] agree, Essence may be seen “[...] as a way to describe the scope of a curriculum, and the scope of each course.”

In such a setting the first two phases of the *Integrated Approach* may provide support early in the curriculum and later on at conducting a course project.

4.5 ESSENCE KERNEL PUZZLER

The *Essence Kernel Puzzler*, from now on short *Puzzler*, was designed as markedly low-threshold offer to get familiar with Essence kernel’s vocab and some of its concepts. With just choosing a nickname everyone can use the Puzzler online¹³.

When discussing various concepts in SE, it is important that people share equivalent ideas when using the same vocab. Discussing Essence repeatedly reveals that people who are in the SE (education) business for years all have their own slightly varying terms and definitions of various concepts. Students of SE without years of SE experience are less likely bound to terms but have more likely a more or less fuzzy idea of different terms.

The goal of the Essence Kernel Puzzler is to make people entering the Essence world familiar with definitions of terms used in the Essence kernel vocab. Of course, everyone could download and read the (in most parts nice to read) Essence specification[188]—and should at some point in time—but a practitioner or student might be slightly deterred or overwhelmed by a document weighing some hundred pages.

Nowadays the Essence community goes to great lengths to make the entry into Essence more bite-sized at the SEMAT homepage[245] and offers differently focused entry points for various target groups. The Essence Kernel Puzzler supports those efforts by making the learning of kernel’s elements definitions and thinking about relation-

¹³ The Essence Kernel Puzzler is available at <https://puzzler.sim4seed.org>

ships between them (hopefully) more accessible and slightly more enjoyable.

In the Integrated Approach presented in this thesis, the *Puzzler* supports the first phase where students get familiar with basic concepts.

4.5.1 *Learning Objectives*

The *Puzzler* focuses on Alphas and their associations and introduces kernel's Activity Spaces as well as the Competencies defined by the Essence kernel.

The *Puzzler* starts with introducing the three areas of concern and proceeds with the seven Alphas and their cards. The following three levels focus on the associations between the Alphas looking at them from various angles. To get a holistic view, these associations play an important role to think about interdependencies demanding for a balanced progressing of all alphas. Two following levels pay attention to ActivitySpaces, their cards and position in one of the three areas of concern. The *Puzzler* concludes with a level about Competencies and their cards. After mastering the *Puzzler* students should be able to name elements of the Essence Kernel, to assign definitions correctly and to establish connections between kernel's elements. Mapping the learning objectives to the cognitive and knowledge dimensions of the Revised Bloom's Taxonomy it targets lower cognitive levels, esp. the first levels *factual* and *remember*.

Mapped to the learning game classification of Dondi and Moretti[74] the *Puzzler* aims at the learning objective to memorize factual knowledge and should provide an increasing level of difficulty, a constraining time factor, and a low level variation of game set.

4.5.2 *Target Audience / Player Profile*

The *Essence Kernel Puzzler* addresses students who already got introduced into the goals of SEMAT Essence. Ideally, they already got shortly introduced to Essence, e.g. by a case study showing the practical use of Alphas and their cards to assess the state of an endeavor at project's kickstart[244].

4.5.3 *Concept and User Interface Overview*

The *Puzzler* is provided with a simplistic user interface. Each level gets introduced with some short basic information about the topic to master. Each level presents a task to handle via dragging and dropping elements to their correct position, e.g. a name of an Alpha to its corresponding card. The time needed to master the level is measured

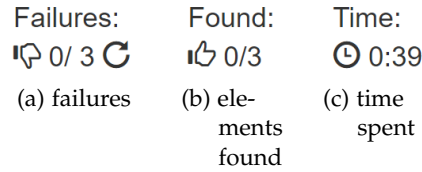


Figure 4.2: Screenshots of Puzzler's status bar Elements

and provides motivation to solve the puzzle faster once repeated (cf. figure 4.2c). The *Puzzler* gives feedback about the progress by counting the number of correctly assigned elements (cf. figure 4.2b) and failed attempts (cf. figure 4.2a). If a player fails too often, the progress of the current level gets lost, and the level starts again. This way unconscious trial and error acting gets not rewarded.

4.5.4 Levels

The *Puzzler* is organized into eight levels. Table 4.5 on the next page and figure 4.3 on page 211 give an overview of the topics addressed and the assumed level of difficulty.

4.5.5 Results

As already mentioned the *Puzzler* was designed a low-threshold offer inviting to get started in no time. That is why intentionally no registration is required to start playing. To collect measures about the usage of the game even so, at each new usage of the *Puzzler*, signaled by changing the nickname at the start of the game or by the first visit of *Puzzler's* website, a unique token gets created and stored to identify this new instance of the game.

This approach has some drawbacks when it comes to analysis of the measured results. Under certain circumstances, it is possible that a new player continues a game started by another player or that a recurring player does not get identified correctly and is counted as a new player again. Therefore results of statistical analysis have to be treated with caution.

So far¹⁴ 76 game instances and 788 attempts to master levels of the *Puzzler* were counted, of that 280 (35.5%) failed and 508 (64.5%) were successful. Players spent in sum 22 hours trying to master the levels of the *Puzzler*. On average each level of the *Puzzler* was played 98.5 times (sd 58.6, median 88), cf. table 4.4 on page 212). Not all of the game instances (n=76) resulted in a mastered first level (n=69). While

¹⁴ at this writing in mid-July 2016

#	Title	Level Mission	Assumed Level of Difficulty
1	Three Areas of Concern	identify the names of the 3 Areas of Concern by choosing them out of a set mixed with Alpha names	*
2	7 Alphas and Their Cards	thinking about Alphas' definitions, drag the correct Alpha to its card containing Alpha's definition	**
3	Alphas' Journey	thinking about the associations of kernel's Alphas, drag the Alphas to their correct position in a figure presenting all Alphas and their given associations	***
4	Connect All the 7 Alphas	thinking about the associations of kernel's Alphas, drag the Alpha Associations to their correct position in a figure presenting all given Alphas and their associations	****
5	A Pair of Alphas	thinking about the associations of kernel's Alphas, choose the correct two alphas that are connected by a given alpha association	*****
6	Spaces for Activities	thinking about kernel's ActivitySpaces, drag the correct ActivitySpace to its card containing ActivitySpace's definition	**
7	Spaces for Activities II	thinking about kernel's ActivitySpaces, drag all 15 ActivitySpaces to their correct position	****
8	It's About Competencies	thinking about kernel's Competencies, drag the correct Competency to its card containing ActivitySpace's definition	*

Table 4.5: Short Description of Puzzler's Levels

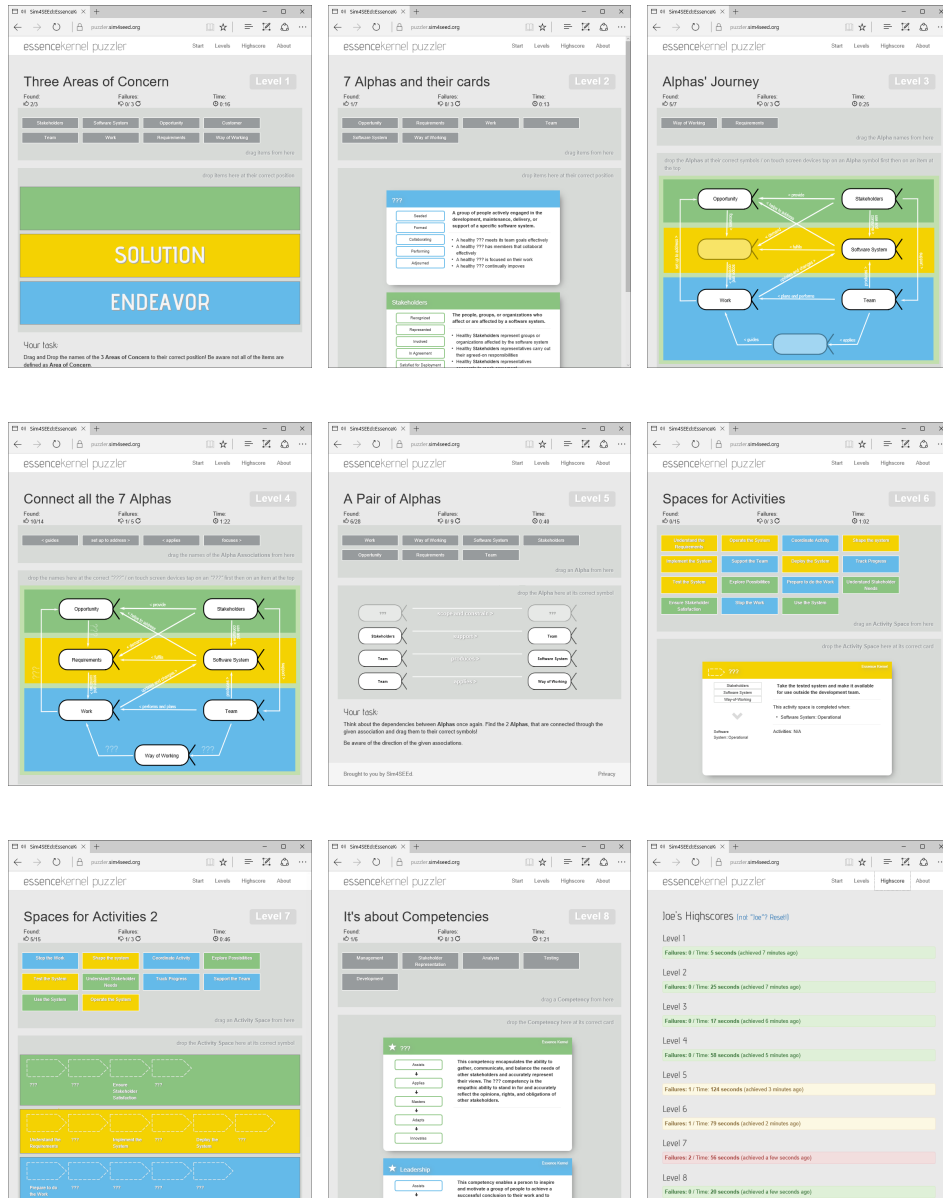


Figure 4.3: Screenshots of Puzzler's Levels and Highscore

a remarkable part of the players masters each level just once, others repeat to master the levels multiple times, which might be motivated by the wish to improve the time needed or to decrease failures made (cf. table 4.5 on the facing page).

Overall the number of players mastering Puzzler's levels is decreasing from level to level. Starting with 69 players mastering level 1 only 21 were mastering level 8. To some extent the exit of players seems to correspond with the assumed difficulty of single levels. Besides the biggest drop after level 1 the levels 4 and 5, which are assumed to be harder than the starting ones, are related to remarkable player exits. Players who made it to level 6 seem to be willing to complete the levels remaining.

The time needed to master or fail a level corresponds to its assumed level of difficulty (cf. table 4.5 on page 210 and figures 4.7 on page 215 and 4.8 on page 216).

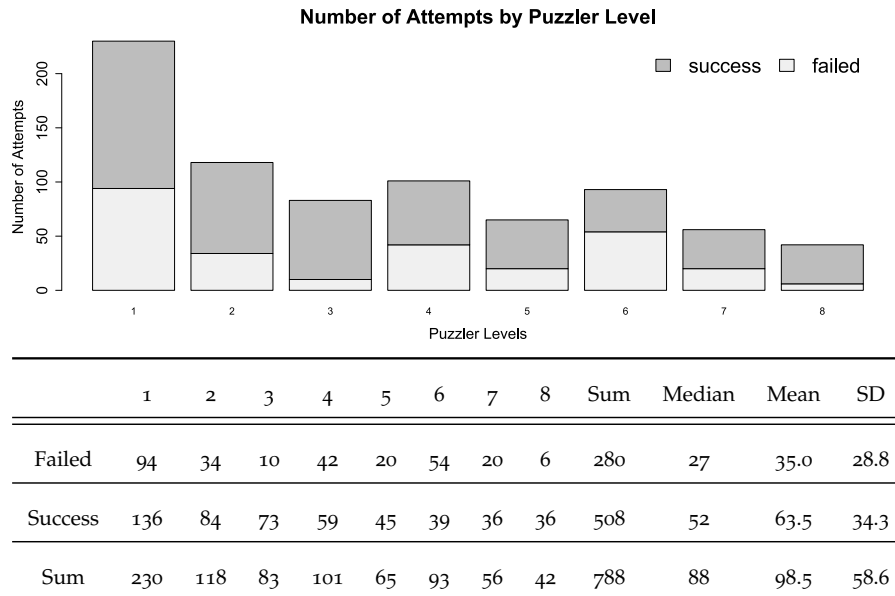


Figure 4.4: Total Number of Attempts by Puzzler Level

Taking the setup into account and analyzing *Puzzler's* measures it is hard to evaluate to what amount the *Puzzler* provides a learning effect. If a level got mastered in the *Puzzler*, a knowledge acquisition hardly could be attributed to the *Puzzler* (alone). Only an experiment in a controlled environment with pre- and post-test and ideally a control group could provide such claims and enable deeper analysis.

But nonetheless, it can be attributed to the *Puzzler* that students dealt with Essence topics. Of the 280 failed attempts to master a level of the *Puzzler*, 83.9% (n=235) were mastered afterward. These numbers are encouraging.

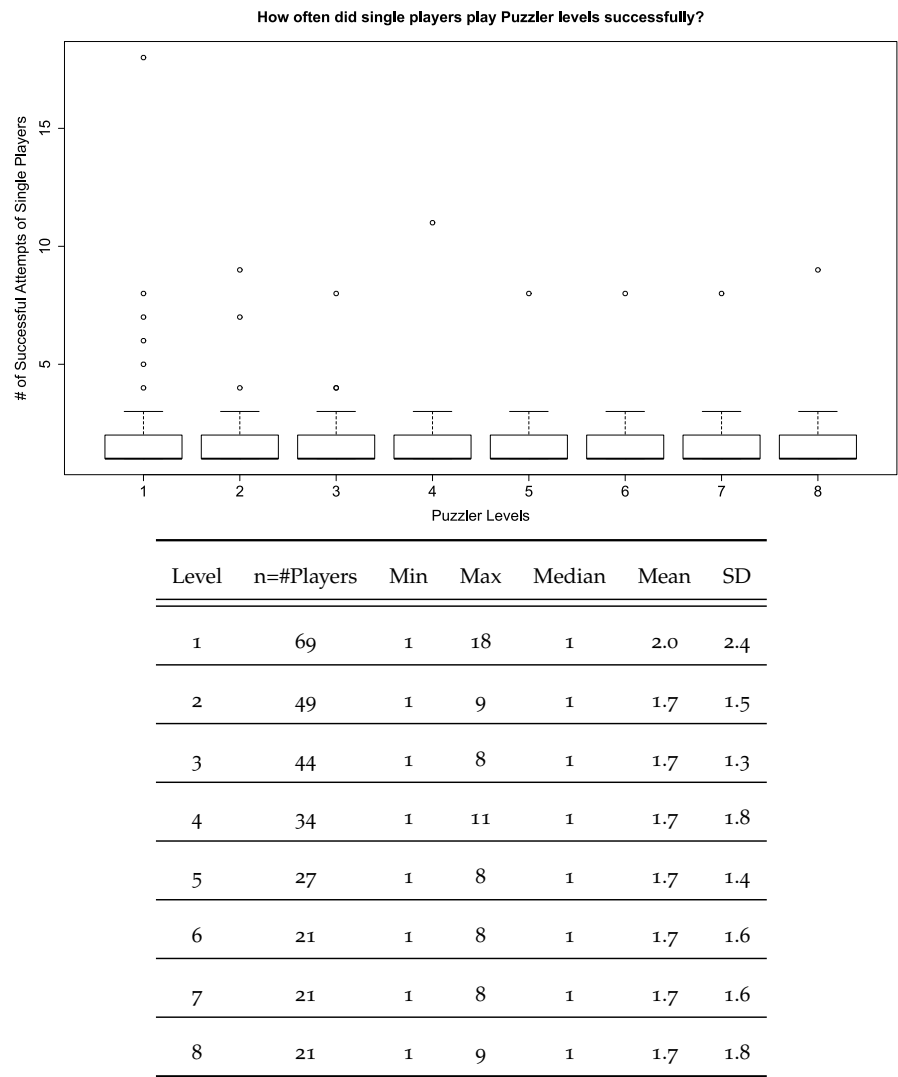


Figure 4.5: Successful Level Attempts of Single Players

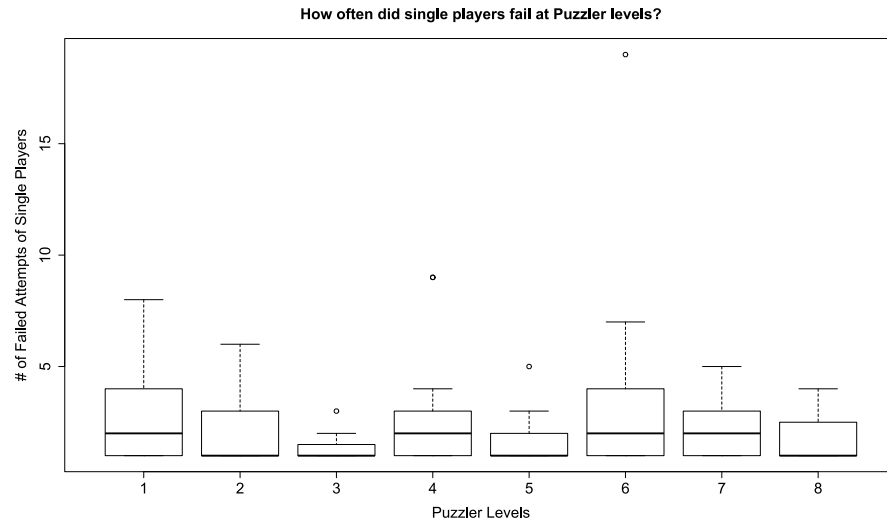


Figure 4.6: Failed Level Attempts of Single Players

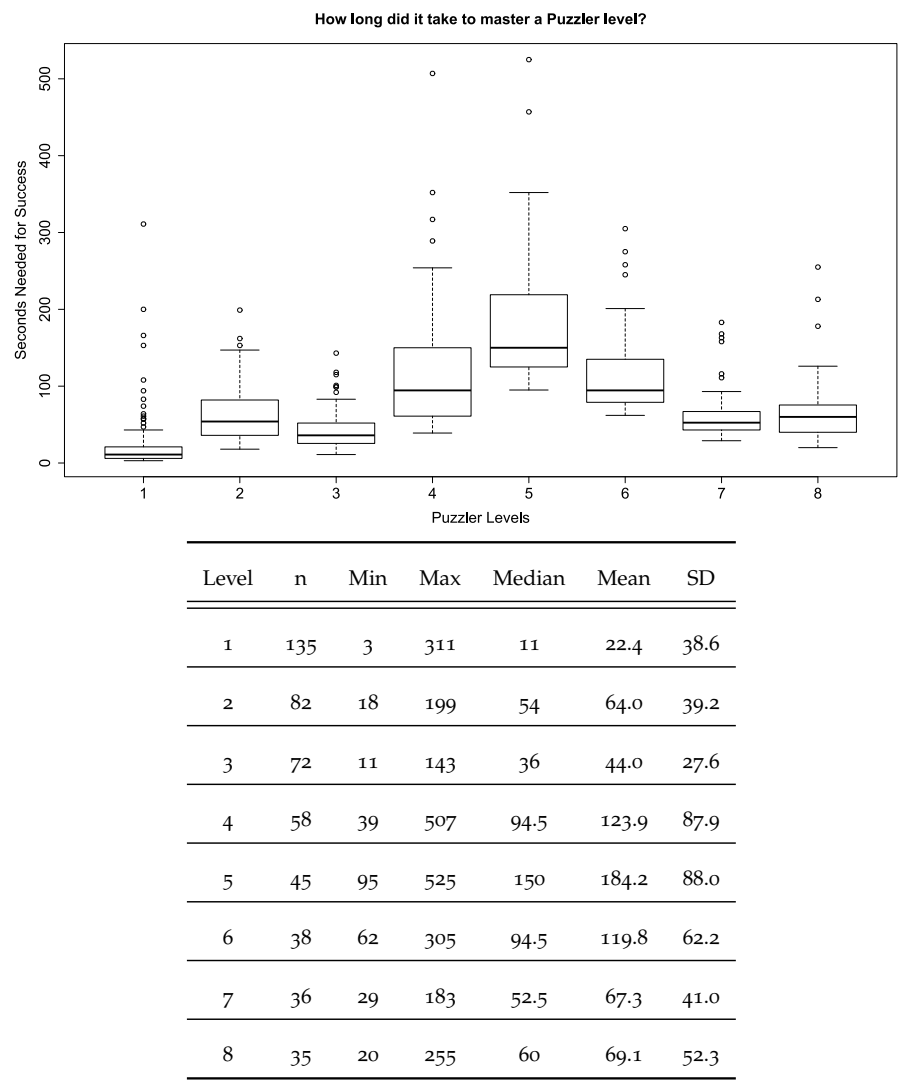


Figure 4.7: Seconds Needed to Master Puzzler Levels (outliers $(x|x \in X, x > \mu_X + 4\sigma_X)$ removed)

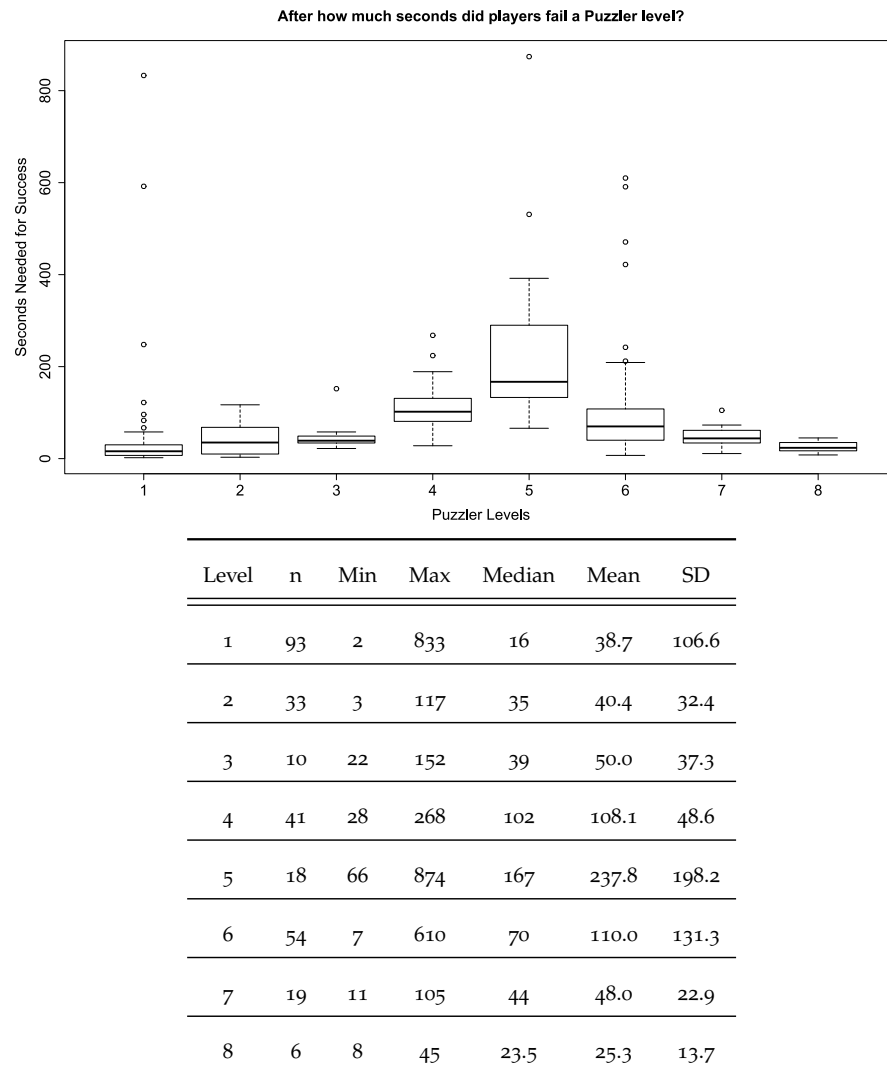


Figure 4.8: Seconds Before Failing a Puzzler Level (outliers $(x|x \in X, x > \mu_X + 4\sigma_X)$ removed)

Results of a questionnaire conducted after the *Simulation Game* reveal that students who used the *Puzzler* acknowledge it as a tool to get familiar with Essence kernel's vocab and concepts. They indicated that they would use it again to recap Essence kernel vocab and recommend it to friends and fellow students (cf. section 5.2.1 on page 258).

4.6 ESSENCE NAVIGATOR

The Essence Navigator represents an elementary execution environment for Essence kernels and methods¹⁵ as defined by the operational semantics of the Essence specification [188]. It provides functionality for holding a level 1 model of an Essence kernel or method, creating and populating the level 0 model to track the overall state of the SE endeavor, to determine the current overall state of the endeavor and to give elementary advice about next steps to do.

It does not aim at fulfilling all of the operational semantics of the Essence specification since it does not need to fulfill its purpose in the presented Integrated Approach. Instead, it focuses on the features needed to support the approach.

In the *Integrated Approach* presented in this thesis, the Navigator connects *Phase 2* and *Phase 3* by providing its unique features to the *Simulation Game* and the course project environment.

4.6.1 Guiding Principles

The following principles guided the design and implementation of the Essence Navigator.

4.6.1.1 Focus and Simplicity First

The Essence Navigator has to support first steps in using Essence concepts and driving SE endeavors. For that reason it should not need too much time to get familiar with it, once a user knows about basic concepts of Essence and the Essence kernel. For the intended use in the *Integrated Approach*, it should not overwhelm the user with very detailed documentation available in the Essence specification but deliver an intuitive starting point for exploration of the concepts.

4.6.1.2 Embeddable Into Simulation Game And Course Project Activities

To support the Integrated Approach presented in this thesis, the Navigator acts as connecting link between the *Simulation Game* (Phase 2) and the course or capstone project of students (Phase 3). Using the Navigator inside the Simulation Game the same way as in a real project it provides students with assurance and familiarity that the

¹⁵ Currently it actually supports only the SEMAT Essence kernel (cf. section 4.6.5 on page 225 for details).

tool and the underlying concepts are of great value when it comes to driving an SE endeavor with multiple dimensions and lots of inflowing information and demands that need a structured and goal-oriented proceeding. To fully support the approach, the Navigator has to be seamlessly integrable into the Simulation Game providing its features as part of the gameplay (cf. section 4.7.3 on page 231).

4.6.1.3 *Embody All Elements of the SEMAT Essence Kernel*

Some of existing approaches to integrating the Essence kernel and its concepts into SE education focus solely on Alphas and disregard ActivitySpaces. The approach presented here favors a tight integration of ActivitySpaces into the learning experience. ActivitySpaces are the generalized abstract placeholders for concrete Activities hence provide the perspective of the things that are always to do. They are a valuable point of integration with practices delivering concrete activities and provide an important categorization useful to classify concrete activities in software processes a student might get faced with in future. That way they provide valuable contents raising the transferability of knowledge gained by using the presented approach.

4.6.1.4 *Extensibility*

While the work on the Navigator was focused on the features needed to support the presented *Integrated Approach*, the tool is designated to get developed further by future work. Right from the start its back-end supports the import of arbitrary Essence methods composed in an external standard tool. Future work might extend the already provided features as needed.

4.6.2 *User Interface Overview*

The following sections describe features and user interface of the Essence Navigator.

4.6.2.1 *Alphas Overview*

The Alphas Overview is the starting point of the Navigator. Figure 4.9 on the next page shows the Alphas overview for an SE endeavor using the SEMAT Essence Kernel. The Alphas are presented with a short summary of their current state. We can see that the endeavor is already quite progressed. *Requirements* are already *Fulfilled*, and the *Software System* is *Usable*. Once the stakeholders confirm their satisfaction that the *Software System* produced addresses the *Opportunity* that Alpha could get progressed to *Addressed*. The team could bring operational support in place and make installation as well as

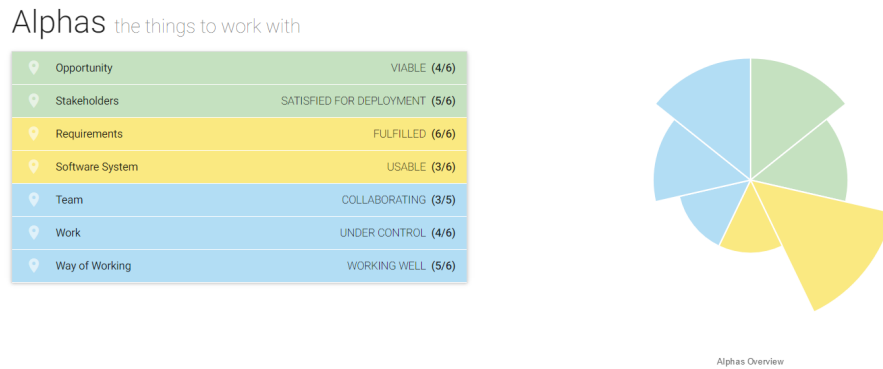


Figure 4.9: Screenshot of the Essence Navigator showing Alphas Overview

user documentation available to make the *Software System Ready* so it could get deployed etc..

The shown endeavor also shows that Alpha *Team* is in state *Collaborating* and not yet in state *Performing*, where it would be ideally in that progressed endeavor. Paying attention to the reason prohibiting the *Team* to unfold its potential fully should be a highly prioritized next task.

4.6.2.2 Alpha Balance Indicator

The Alphas of the Essence Kernel constitute a net of interdependent elements and have to be progressed in a balanced way. The Essence Navigator indicates the balance of the Alphas of the current endeavor via a graphical representation where each Alpha is represented by a slice of a pie growing as its AlphaStates get progressed. Alphas are presented colored by standard color codes of their respective area of concern. Figure 4.10 on the following page shows different manifestations of possible endeavor states. With a single glimpse, even the untrained viewer of the indicator gets a first impression of the (un)balance of the given SE endeavor. The more the indicator is shaping a circle, the more the endeavor is balanced. The more the indicator shapes a fringed structure, the less the Alphas of the endeavor are balanced. The smaller the indicator is presented, the less the Alphas of the endeavor are progressed. The bigger the indicator is presented, the more the Alphas are progressed. An indicator of an endeavor with none of its Alphas progressed to a first AlphaState is not visible. As given by the semantics of Alphas and their AlphaStates an indicator forming a perfect circle might not be striven for at every point in time.

4.6.2.3 Alpha Detail and AlphaState Detail View

At choosing one of the Alphas, it reveals its details. Figure 4.11 on page 221 presents the Alpha (a) and AlphaState (b) detail views of

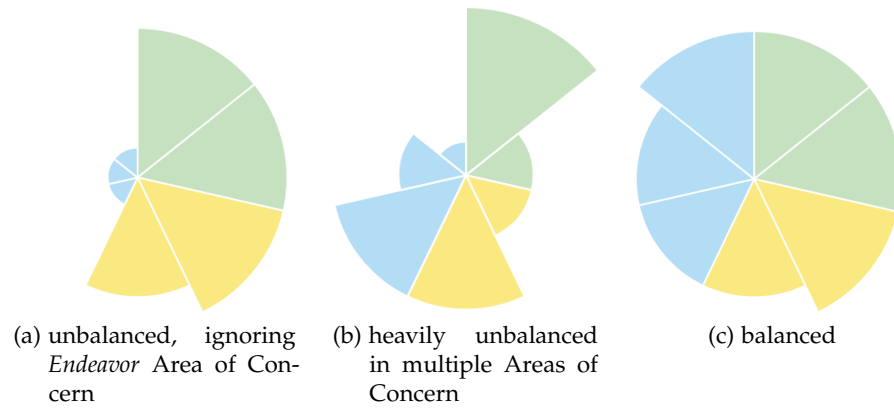


Figure 4.10: Screenshots of the Essence Navigator Showing Alpha Balance Indicators

the *Requirements* Alpha. We can see that all of the AlphaStates are reached. This is indicated by the dark green color and the double-tick symbol. The numbers in parentheses on the right side indicate how much of the Checkpoints of an Alpha are already fulfilled in relation to the total number of Checkpoints of that respective AlphaState.

The AlphaState detail view lists all the Checkpoints given to assess the given AlphaState. The figure shows that there is one Checkpoint missing to progress to AlphaState *Fulfilled*. Below the list of AlphaStates we find links to the respective ActivitySpace(s) addressing the state. In the example given, the ActivitySpace *Test the System* has to be performed to achieve the results needed for a successful Checkpoint assessment.

4.6.2.4 Activities Overview and Details

Following the link to an Activity at the AlphaState detail view, a user arrives at the Activity overview and details view. The overview lists all Activities of the chosen Essence kernel or method. Activities are again colored in the standard color coding of their areas of concern. That way it is easy to establish a connection between the *Alpha Balance Indicator* at the right side and the respective Activities addressing AlphaStates of the Alphas represented.

Figure 4.12 on page 222 shows the detail view of *Explore Possibilities*. This one provides links to all the AlphaStates addressed by that Activity. Following one of the links takes the user to the detail view of the respective AlphaState.

4.6.2.5 Competencies Overview

The Competencies overview and details views present the Competencies defined by the SEMAT Essence kernel. The latter shows the short description and levels defined by the Essence specification. These

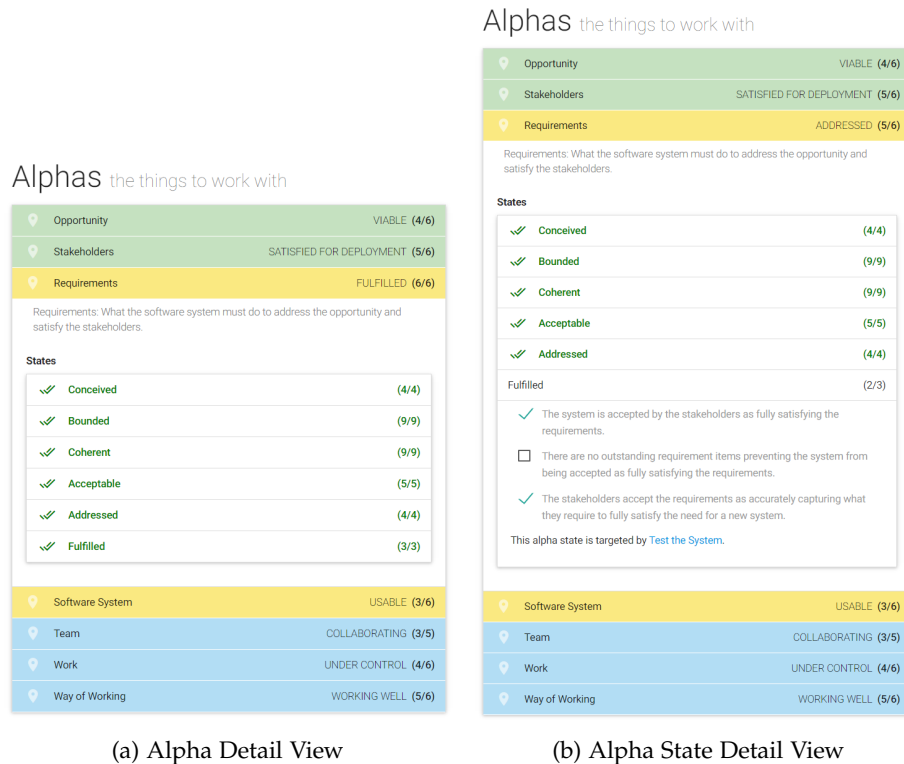


Figure 4.11: Screenshots of the Essence Navigator: Alpha and AlphaState Details of the Requirements Alpha

views do not provide any links since the SEMAT Essence kernel does not define any associations of Competencies and their levels to ActivitySpaces.

4.6.2.6 To-Do List

The To-Do list of the Navigator supports at defining next tasks to do. By processing the current state of the SE endeavor and utilizing the associations between ActivitySpaces and AlphaStates, it provides the next actions that may be taken to progress all of the Alphas to their next AlphaState. In this sense, the list acts as elementary *Guidance Function* as defined by the Essence specification [188]. Obviously, this can be only first suggestions since the Navigator does not know about any particular project needs and specific goals of a team at a given point in time of a running SE endeavor.

4.6.3 Essence Kernel/Method Composition And Enactment

To run an SE endeavor in the Essence Navigator, an Essence kernel or method¹⁶ model needs to be enacted first. The process of enactment

¹⁶ Currently the *Essence Navigator* is capable only of Essence kernel elements, esp. *Alphas*, *Alpha States*, *Checkpoints*, *Competencies*, and *Activity Spaces*. Future work may

Activities the things to do

Explore Possibilities

Explore the possibilities presented by the creation of a new or improved software system. This includes the analysis of the opportunity to be addressed and the identification of the stakeholders.

This activity targets at reaching the following alpha states:

- Stakeholders:-Recognized
- Opportunity:-Identified
- Opportunity:-Solution Needed
- Opportunity:-Value Established

Understand Stakeholder Needs

Ensure Stakeholder Satisfaction

Use the System

Understand the Requirements

Shape the System

Implement the System

Test the System

Deploy the System

Operate the System

Prepare to do the Work

Coordinate Activity

Support the Team

Track Progress

Stop the Work



Alphas Overview

Figure 4.12: Screenshot of the Essence Navigator showing Activities Overview

Competencies the abilities needed

Stakeholder Representation

Analysis

Development

This competency encapsulates the ability to design and program effective software systems following the standards and norms agreed by the team. The development competency is the mental ability to conceive and produce a software system, or one of its elements, for a specific function or end. It enables a team to produce software systems that meet the requirements.

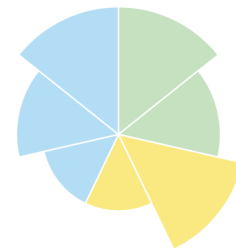
Levels

- 1 - Assists**
Demonstrates a basic understanding of the concepts and can follow instructions
- 2 - Applies**
- 3 - Masters**
- 4 - Adapts**
- 5 - Innovates**

Testing

Management

Leadership



Alphas Overview

Figure 4.13: Screenshot of the Essence Navigator showing Competencies Overview

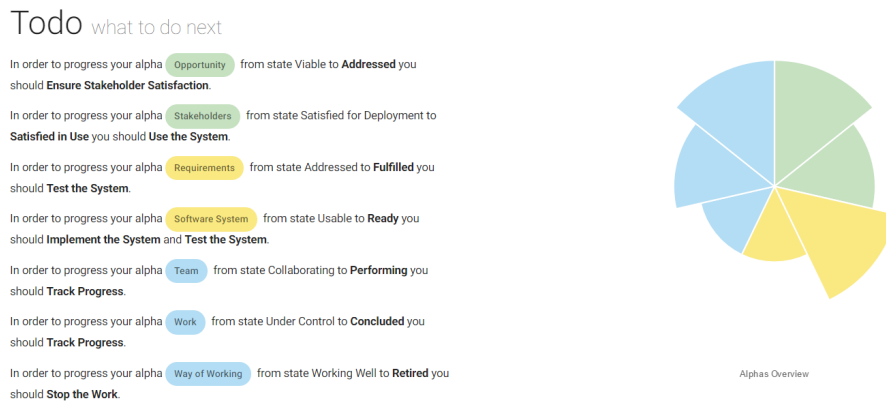


Figure 4.14: Screenshot of the Essence Navigator Showing a To-Do List Based On Endeavor's Current State

includes the generation and population of a level 0 model able to track the progress of the endeavor.

To create a level-0 model, a corresponding level-1 model is needed first. Instead of implementing an own editor for such models the Navigator makes use of a standard tool[115] provided to the Essence community¹⁷ The XML Metadata Interchange (XMI)-output of that standard tool representing an Essence Method Model at level 1 gets parsed and transformed into an object graph representing an Essence Method model at level 1. This object graph gets stored into a Method Store. Each time an SE endeavor gets created in the Navigator Backend the Essence Method at level 1 gets enacted. All needed Essence elements at level 0, e.g. Alphas, AlphaStates, and Checkpoints, get created and populated enabling the Navigator to track the progress of the newly created endeavor.

To accomplish this, an Essence component, implementing the Essence Language, and an XMI-parser were developed. The Essence component and the Navigator were implemented utilizing the Ruby programming language[278]. The frontend and backend of the Navigator are implemented using the Ruby on Rails[279] web development framework and the React[85] JavaScript library.

provide the additional elements to enable the simulation of Essence methods, including elements like *Activities*, *Sub-Alphas*, *WorkProducts*, and their *Levels of Detail*.

¹⁷ Currently supported is only the EssWork Practice Workbench of Ivar Jacobson International (IJI). "IJI is proud to support the SEMAT initiative and to offer its new practice authoring tool "The EssWork Practice Workbench" for use by the SEMAT community for FREE. Based on the SEMAT Method Architecture, EssWork Practice Workbench comes with an interactive version of the Essence Kernel, and enables the creation of kernel extensions and practices as well as their composition into methods." [247]

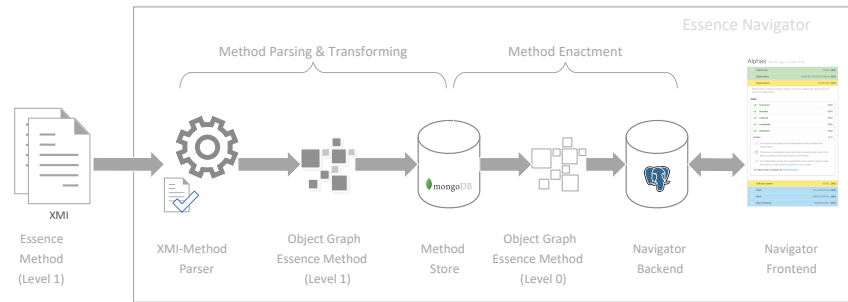


Figure 4.15: Essence Method Parsing and Enactment Schematic Overview

4.6.4 Related Work

As the Essence specification is quite new¹⁸, tool support is in early stages. The Navigator makes use of the *EssWork Practice Workbench*, provided by Ivar Jacobson International (IJI). This tool is an authoring environment providing the opportunity to compose kernels, practices and methods based on Essence and to export these models to static web pages. This tool does not provide the capabilities to enact and run endeavors based on the composed models.

SematAcc[98] is another tool which provides running (“accelerating”) of endeavors based on the SEMAT Essence kernel. It allows to create projects and populate an Essence kernel to track projects’ progress. The tool does not provide the opportunity to edit any of kernel’s elements, e.g. Checkpoints. It provides a static kernel representation. Checkpoints can not be checked themselves. Only AlphaStates can be checked as a whole, which does not allow for a fine-grained assessment of the current state of an endeavor. *SematAcc* provides kind of event logging, documenting the changes of AlphaStates and the point in time of that change. This feature might be used to “generate data for research purposes.”[98]

The *SematAcc* project provides a running online demo version of the tool¹⁹. Figure 4.16 on the next page shows the GUI of *SematAcc*. Some similarities to the GUI of the Navigator are clearly apparent. *SematAcc* does not provide any perspective on ActivitySpaces or Competencies. The relationships between AlphaStates and ActivitySpaces addressing them are not considered, and it does not provide any guidance function to support the identification of next steps to take. It seems that simplicity was one of the guiding principles of the design of the tool too. For the requirements of the *Integrated Ap-*

¹⁸ Version 1.0 of the specification was released in November 2014. Current version 1.1. was released in December 2015.

¹⁹ An online demo version is available at <http://sematacc.herokuapp.com/> (lastly visited on 07/08/2016)

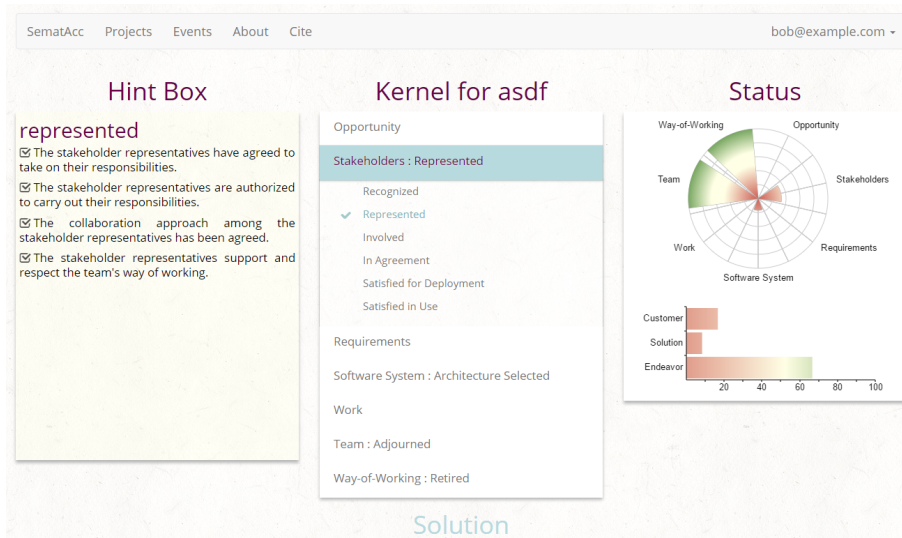


Figure 4.16: Screenshot of the *SematAcc*[98]: GUI

proach presented in this thesis, it is too simple. It neither supplies the features needed nor is it integrable into the *Simulation Game* in the required way.

Essencia of uEngine Solutions is an open source tool based on an open source workflow and BPMN engine that got introduced at the OMG event “Essence in Practice: A Revolution in Software Engineering?”[189, 246] The presentation of the tool was very promising and created the impression of a comprehensive feature set ranging from method definition to method enactment. Unfortunately, it seems that any further information about the tool is very scarce. Despite some UI specification presentations available online only in Korean, no information about the current state of the tool seems to be available. From the first impression given at the OMG event, the tool and its foundations in workflow and BPMN seemed overly complex for the requirements of the *Integrated Approach* presented in this thesis.

4.6.5 Future Work

The development of the Essence Navigator was focused on the features needed for the *Integrated Approach* presented here. Since the Essence kernel was chosen as the foundation of the activities, it currently fully supports only the presentation of the SEMAT Essence kernel and similar kernels and methods. While it can parse and import Essence methods composed in a standard tool to any given level, its frontend capabilities are quite limited yet. These limitations of the tool did not impose any restrictions on the case study evaluating the *Integrated Approach* presented here but might be limiting in scenarios running SE endeavors based on more complex Essence practices and

methods. Features still open to supporting more complex Essence Method models as well as integration with other standard tools of SE teams, e.g. issue tracking systems, are part of future work.

4.7 SIMULATION GAME

From a software design perspective, the *Simulation Game* acts as *Decorator* of the simulation model introduced in chapter 3 on page 117. It provides the features needed for a game, facilitating collaboration as well as competition and supports its usage inside of a course environment running one game for each of the course members.

4.7.1 Learning Objectives

After mastering the *Simulation Game*, students should be able to apply acquired concepts in a simplified defined simulated context. They should feel comfortable with performing Essence's "mechanics" and applying the *PDCA-cycle* (cf. figure 2.16 on page 107). Students should be able to orientate inside a given SE method/kernel and to consider all essential dimensions (Alphas) of the SE endeavor. They should be able to assess the progress and health of their endeavor in a structured and continuous way (at least at the abstracted level of the *Simulation Game*). It would be unrealistic to assume that students would know all the Alphas, their states, and their checklists in detail after playing for about 90 minutes. The game provides the opportunity to get to see the multitude of different information presented to project members in an SE endeavor and to experience how Essence supports the team in structuring this information—how Essence is guiding the team through the project in a holistic way. Mapping the learning objectives to the *cognitive* and *knowledge* dimensions of the *Revised Bloom's Taxonomy* (cf. section 2.3.1 on page 38), the *Simulation Game* targets the cognitive levels *remember*, *understand*, and *apply*. All levels of the knowledge dimension (*factual*, *conceptual*, *procedural*, and *metacognitive*) get addressed. Furthermore, it is an objective of the *Simulation Game* to support students in developing an *attitude* that lets them apply their skills and knowledge—by providing the experience of support and guidance given by Essence concepts in the dynamic context of an SE endeavor (cf. section 4.1.3.1 on page 186).

4.7.1.1 Target Audience / Player Profile

The *Simulation Game* addresses students already introduced into the goals of the SEMAT Essence Kernel and familiar with basic concepts like Alphas, AlphaStates, and their Checkpoints as well as ActivitySpaces. Students ideally got already introduced to the assessment of the state of an SE endeavor, e.g. by a case study showing the practical

use of Alphas, AlphaStates and their cards to assess the state of an endeavor at project's kickstart[244].

4.7.2 Guiding Principles

The design and implementation of the Simulation Game were driven by the guiding principles described in the following sections.

4.7.2.1 Complementing and Preparing Real Project Work

Time is a very limited resource in every curriculum. The proposed approach was not designed to replace real SE project work in the form of course/capstone projects. It should not just raise time on task of students, particularly not take away more time than necessary from real SE project work in form of course or capstone projects. Instead, it should give students a quick and profound start in such activities, provide already familiar concepts and tool support hence raise conscious goal-oriented attitude and lower the cognitive load students are faced with in such projects. Playing the *Simulation Game* based on the Essence kernel should be achievable in a standard course unit of 90 minutes length and supplemented by debriefing activities (cf. section 4.7.9 on page 244). Students may extend the time spent with the game outside of regular course units or return to it throughout the duration of the course to compare their game results with their real project work. The virtual time provided inside the game depends on the budget that was allocated to the players by provided game parameters at game initialization (cf. table 3.13 on page 173).

Focused on Software Engineering Essentials

Existing approaches of games supporting software process education pay quite a lot attention to activities like choosing the one team member who fits best to accomplish a task. This is not a focus of the presented Simulation Game—for several reasons.

To repeatedly emphasize, that only an expert with years of experience in specific disciplines can accomplish the results required, might be contra-productive and less motivating for yet less experienced students facing their own upcoming SE endeavor and expected to deliver similar results to some extent.

With the wide acceptance and distribution of agile and lean approaches[297], traditional push model workflows, where tasks get assigned to the queue of individual team members, are replaced by pull models, where tasks are hold in a single prioritized common queue and get pulled off the front by team members as they become available. Scrum[140] and Kanban[8] are two popular examples of these agile and lean approaches[297].

The SEMAT Essence Kernel was chosen as a compact starting point to experiment and evaluate the approach. The kernel does not consider single team members but treats the team as a whole, depicted by the *Team Alpha*²⁰. In the *Simulation Game* activities are assigned to the whole team, assuming that members follow a pull workflow, organize them self and manage to accomplish the tasks. Team's possible workload is implemented as a fixed amount of person hours per working day. Activities may be prioritized by allocating different amounts of these available person-hours to activities.

Since the time assigned to playing the game is limited by intention, the game can not impart all known phenomena and effects that might appear at driving an SE endeavor. The kernel encloses the essential aspects common to all SE endeavors but not every detail and aspect that might be of interest only under certain circumstances. By integrating the kernel, the *Simulation Game* follows that approach. Topics and aspects omitted or just broached might be deepened and discussed in debriefing activities or at the following course project itself.

4.7.2.2 *Reward of good SE practice*

SE is defined as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software [...]”[112]. The game rewards good SE practice and penalizes bad ones. The point system and the ranking system assure that the chance to get points by accident should be minor. A player getting points just after a sequence of unconscious try and error decisions should not gain an advantage over a conscious player attempting to find proper decisions (cf. section 4.7.4 on page 237).

4.7.2.3 *Facilitating Interaction and Discussion In Real World*

To facilitate reflective analysis and social interaction, the game provides occasions for discussion. It delivers feedback indicating how well player's decisions were compared to the decisions of other team members—and how well the whole team performs compared to other teams in the course. To support debriefing activities, the game collects results of all players and their teams.

4.7.2.4 *Supporting Acquisition of new Knowledge*

“Yet it seems that, in general, game-based learning appears to have more impact on lower cognitive levels, reinforcing knowledge learned earlier. It seems inadequate for teaching new knowledge.”[301] It is the objective of *Phase 1* (cf. section 4.3.1 on page 202) to learn and reinforce concepts and vocab, overall addressing lower cognitive

²⁰ The *Management Extension*, provided by the specification [188] to complement the kernel, contains the sub-Alpha *Team Member* but was not chosen as the foundation of the first version of the *Simulation Game*.

levels (4.3 on page 188). It is the objective of *Phase 2* to let players actively *apply* concepts, to *analyze* and *reflect* on their application in a PDCA cycle (cf. section 2.16 on page 107). The *Simulation Game* supports debriefing activities, where the results of the whole course get *analyzed* and *evaluated*, preparing students for their own real SE endeavors (*Phase 3* of the Integrated Approach, cf. section 4.3.3 on page 205).

4.7.2.5 Supporting Knowledge Transfer

By choosing the Essence kernel as the foundation of the *Simulation Game*, students get familiar with a compact thinking framework which is not bound to any practice or software process and supports driving any SE endeavor. By integrating the fundamental mechanisms of endeavor assessment, the definition of next goals, choosing the next appropriate activities to accomplish these goals and monitoring the progress and health of all essential dimensions directly into the gameplay, students acquire highly transferable knowledge and are getting ready for action.

4.7.2.6 Appropriate Game Genres And Beneficial Features

Not all game types are good for all learning outcomes[40, 41, 295]. Dondi and Moretti (2007) propose a methodological approach to select learning games and present features required to accomplish chosen learning objectives (cf. table 4.6 on the next page). Following this classification *simulation games*, *adventure games* and to some extent *drill and practice* as well as *puzzle games* and their characteristic features contribute to the defined learning objectives. The presented Simulation Game will provide features listed in table 4.6.

4.7.2.7 Game Visuals

Prensky states “[...] creating engagement is not about those fancy, expensive graphics but rather about ideas. Sure, today’s video games have the best graphics ever, but kids’ long-term engagement in a game depends much less on what they see than on what they do and learn. In gamer terms, ‘gameplay’ trumps ‘eyecandy’ any day of the week.”[218]

With limited resources at the development of the game, the *Simulation Game* will provide a simplistic non-distracting user interface oriented towards instant messengers, requiring no further introduction²¹. By subtly utilizing colors defined in the Essence specification

²¹ Students today are used to using instant messengers, e.g. WhatsApp[266]. WhatsApp, acquired by facebook for 19 billion U.S. dollars in February 2014, is a cross-platform instant messaging service and one of the most popular mobile apps worldwide. It has a more than 1 billion monthly active users worldwide. [303] In Germany 90% of young people aged between 16 and 29 are using WhatsApp. [302]

Learning Objective (Definition)	Features Required (selection)	Appropriate Games (#of Players)
Memory (factual knowledge)	(a) Increasing level of difficulty, (b) Low level variation of the game set and situations	Drill & Practice, Puzzle games (one)
Applying concepts (use information, methods, concepts in new situations)	(c) Presence of a set of rules and instructions both well defined and easily understood, (d) Balance between reality and abstraction, (e) Turn-based games	Drill & Practice (one)
Decision making (strategy and problem solving)	(f) Game situation divided into scenarios with specific goals relatively brief to reach, (g) Availability of documents that describe the situation in a detailed way, (h) Accurate description of the problem, (i) Real-time monitoring of the other player/opponents [...], (j) Background knowledge of content is vital to successful completion or victory	Strategic games, Adventure games, Role play games, Simulation games (one in connection with other players or opponents)
Social Interaction (understanding social environment of others)	(k) Luck does not play a part, (l) Reflection is a factor, (m) Persistent-state game, (n) Presence of tools for communicating [...] with other players, (o) Game completion time is not particularly relevant, Time factor is not a constraint, (p) Feedback is relevant and detailed, (q) At the end of the game the player can review his own strategy	Strategic games, Role play games, Simulation games (Many players)
Ability to learn and self assessment (Evaluation)	(r) Availability of evaluation tools, (s) Availability of relevant documentation (t) Presence of tracking tools and the facility to review previous steps (u) Presentation and Review of the result achieved, (v) Questions to foster reflection, (w) Highlights player's points of strength and weakness, (x) The ability to learn is developed by increasing levels of difficulty and availability of different choices, (y) No place for luck, (z) Good balance between action and reflection	Role play games, Simulation games (one)

Table 4.6: Game-Based Learning for Universities and Life Long Learning classification[74] (selection)

players get supported at following messages sent from the virtual team.

The Essence Navigator, introduced as a tool to support driving real SE endeavors (cf. section 4.6 on page 217) and designed for ease of use, gets deeply integrated into the gameplay and provides realism to players.

By designing and providing a rather lean interface with scaffolding where needed both in the games as well as in the *Essence Navigator* the extrinsic cognitive load (cf section 2.2.5.1 on page 31) is kept down allowing learners to focus on the essential learning content.

4.7.2.8 Evaluation Considerations

A “[...] learning game should be a ‘good game’ through which the player will achieve the stated learning objectives.”[74] To evaluate the approach a combination of heuristics, statistically analyzed questionnaire results and data collected with in-game measurements is utilized. Observations made at conducting the case study complement these results.

HEURISTICS Dondi and Moretti (2007) propose a methodological approach to assess the quality of learning games. They provide an *evaluation framework for assessing games*, which is based on a research initiative that involved organizations from different European countries and expertise. This framework enables the (self-)evaluation of games for learning and aims at an increased quality awareness In appendix C on page 361 the approaches and design decisions taken are mapped to the quality criteria of the *Sig-Glue* quality criteria [74].

QUESTIONNAIRE A questionnaire collecting feedback of experiment’s participants got developed. Its design gets described in section 5.1.6 on page 256. Results of the questionnaire are presented in section 5.2 on page 258.

IN-GAME MEASUREMENT To complement sources of evaluation data, a set of in-game measurements is taken. With its architecture based on web technologies, the *Simulation Game* provides the opportunity to take measures of interest supporting game evaluation.

4.7.3 Gameplay and User Interface Overview

The game uses the setup of the case study “Kickstarting a Project”[244] as starting point of the game. Players slip into the role of a coach directing a virtual team of 3 developers through an SE endeavor having a fixed budget and a fixed cost rate per day.

Figure 4.17 on the next page shows the user interface of the *Simulation Game*. At the left side, we can see the *Essence Navigator* (1)

“A game is a series of interesting choices.”
—Sid Meier

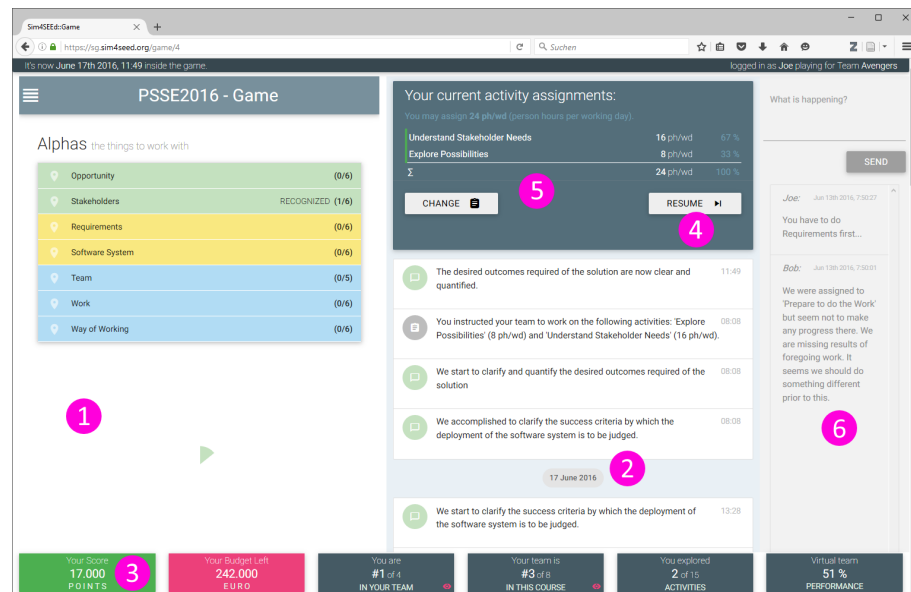


Figure 4.17: Screenshot of Simulation Game: UI Overview

introduced in section 4.6 on page 217. The *Navigator* provides the same features as in a real software project and adds some game specific features when employed inside the game environment. Like in a real project, the *Navigator* facilitates the assessment of the endeavor (cf. figure 4.18 on the next page) and gives guidance at the definition of next steps to take.

In the center of the UI all messages from the virtual team, ordered by calendar day, flow in (2). At the bottom (3) a status bar informs the player about points, budget left, team and course ranking, the number of explored activities and the current performance of the virtual team.

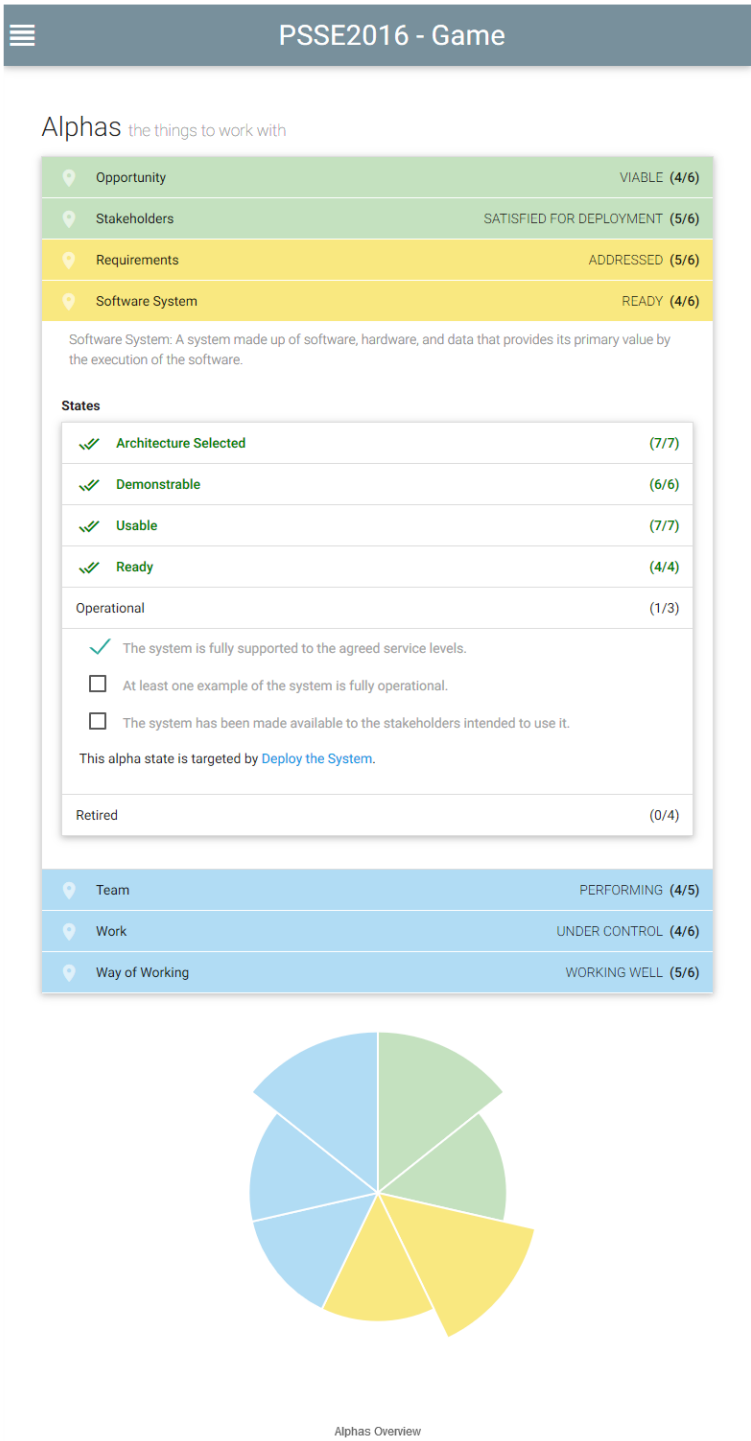
At the top center, a player finds the *Resume* button (4). The *Simulation Game* was designed as a turn-based game giving a player as much time as needed to analyze a situation, assess the endeavor, think about next steps to take, and to take action as desired. Only after clicking the *Resume* button the game continues.

On the right, a player finds an elementary chat providing the opportunity to communicate with (real) teammates (cf. section 4.7.5 on page 239).

The top center area (5) summarizes the currently assigned activities. This is illustrated in more detail in figure 4.21b on page 235.

By assigning activities to their virtual team, the endeavor gets progressed as far as all preconditions for performing the assigned activities are satisfied.

Before a player may assign an activity to her virtual team, she has to explore that activity first. For that, a player makes use of the *Navigator* (cf. section 4.6 on page 217) integrated into the game UI. Inside a game environment, it provides the opportunity to explore and col-



lect Activities. Figure 4.20 on the next page shows details of the UI. Once a player added the respective activity to the *explored activities*, it gets assignable via the assignment form illustrated in figure 4.21a on the facing page. The assignment form appears once a player clicks the *Change* button at the top center of the game UI.

When an activity got assigned to the virtual team, it depends on the fact, if all of the prerequisites to successfully perform the activity, are met (cf. section 3.3 on page 120). If this is the case, the team delivers feedback about their current tasks. Feedback modeled for certain events of Checkpoints gets delivered as messages from the virtual team. This is illustrated in figure 4.22 on the facing page. The list of messages grows while the game progresses and serves as kind of log, enabling the analysis of decisions taken by just scrolling down the list of messages.

Depending on the configuration of the underlying simulation model, the virtual team might send messages at starting work on a Checkpoint to achieve the results required by its description, at its accomplishment, and several other events (cf. 3.7.4 on page 175). Each of the messages from the virtual team corresponds to a Checkpoint. A player then has to analyze that incoming message.

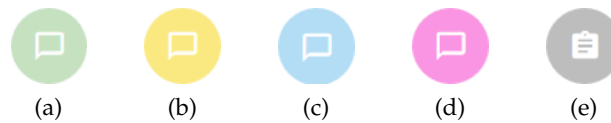


Figure 4.19: Screenshots of Simulation Game: Color Coding of Messages

To simplify the analysis of game messages, each message type was assigned a specific color coding. Figure 4.19 presents a summary of the color codes used. Messages from the virtual team regarding events of Checkpoint processing were colored with standard colors of their respective area of concern. Checkpoints of the *Customer* area of concern are colored green (a), those of the *Solution* area of concern are colored yellow (b) and those of the *Endeavor* area of concern are colored blue (c). To make tutor messages stand out, they are colored pink (d). Standard confirmation messages of the game are colored gray (e).

If the message just expresses that the team *starts* to do something, no action has to be taken by the player. Otherwise, the analysis of a message from the virtual team might trigger a new assessment of an AlphaState. The player should utilize the provided *Navigator* to get support at this point. The player has to find the Checkpoint corresponding to the incoming message and decide if it has to be checked. The *Navigator* acts inside the game as it would in a real SE endeavor.

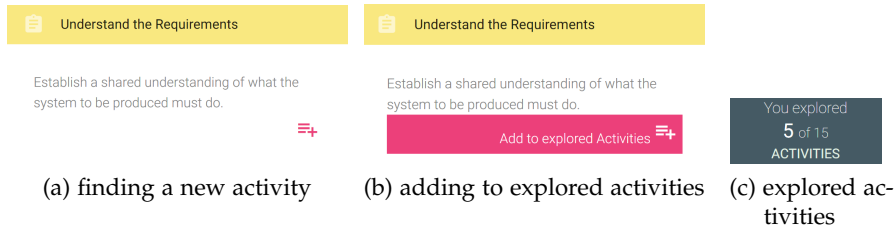


Figure 4.20: Screenshots of Simulation Game: Exploration of Activities Supported By Navigator

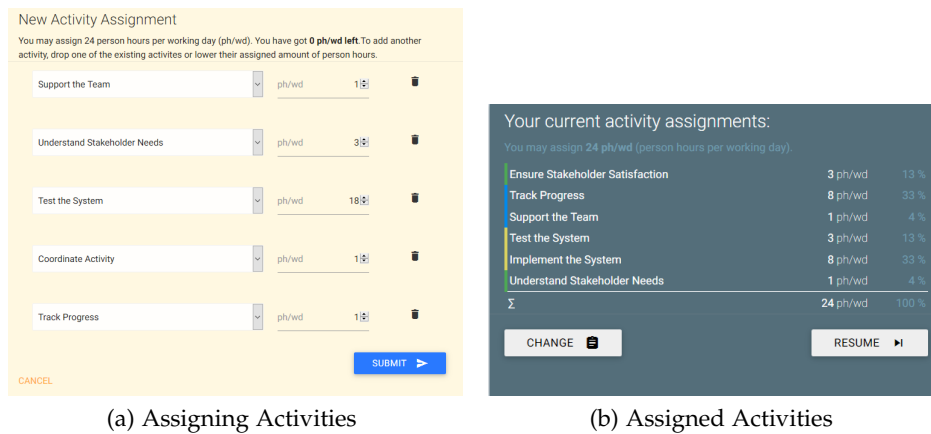


Figure 4.21: Screenshots of Simulation Game: Assigning Activities To the Virtual Team

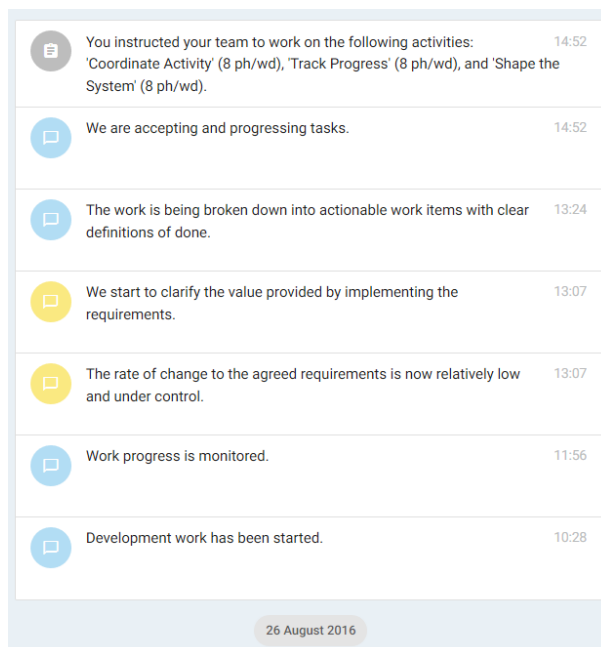


Figure 4.22: Screenshot of Simulation Game: Incoming Messages of Virtual Team

It does not *define* the state of the virtual endeavor but gives support to assess it (cf. 5.2.2.1 on page 262).

To a player consciously processing all incoming information, the *Navigator* perfectly provides the current state of his virtual endeavor—just like the *Navigator* would in an ideal situation in a real SE endeavor. If a player does not process the incoming information in a proper way, he runs into danger to make wrong decisions at assigning activities based on a misconceived impression of endeavors current state—just like in a real SE endeavor.

If all Checkpoints of an AlphaState got fulfilled, a new AlphaState is reached. In that case, the player has to decide if a new activity should be assigned to the virtual team. The *Navigator* supports at this point by linking all the Activities addressing the next targeted AlphaState and by providing an elementary To-Do list.

Should the player assign an activity with open preconditions on its addressed AlphaStates and/or Checkpoints (cf. 3.3), a tutor message gets sent to the player. Figure 4.31b shows such a message, stating that the player assigned two activities to the team but the team can not make any progress because of missing prior work. If the player assigns only activities that cannot be progressed a tutor message gets sent, stating that no progress for a number of days was made at all (cf. section 3.5.3).

Because the cost rate per working day is fixed, a player would have spent money without achieving any progress on the endeavor and might regret decisions made. To enable *learning from failures* the game provides a time travel feature (cf. figure 4.23 on the facing page) enabling the player to travel back in time and make decisions once again—now with increased knowledge. To avoid an abuse of the time travel feature and not to promote unconscious try-error-time-travel cycles, this feature is burdened with cost, making it more attractive to try finding good decisions and to use a time travel only in case of need.

The virtual team of a player starts with a low performance (cf. figure 4.26, at the far right of the status bar). This reflects the assumption that a new team starting an endeavor does not start with full performance.[287] To unfold its full potential, it has to organize itself and its work, find their ideal way of working, has to get support and track its progress. In Essence, all these issues are implemented by the Alphas of the *Endeavor* area of concern. To raise the performance of the virtual team, the Alphas of this area of concern have to be progressed (cf. section 3.5.3 on page 163).

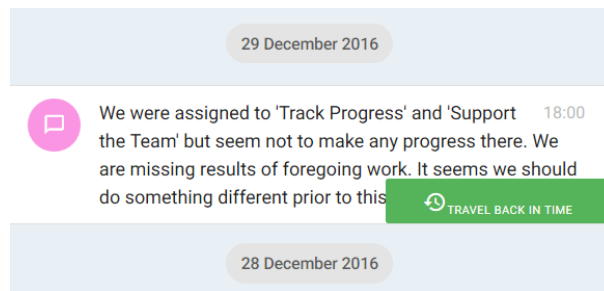


Figure 4.23: Screenshot of Simulation Game: Time Travel

4.7.4 Scoring, Ranking, Leaderboards

Scoring rewarding proper practice

To get a score, the game provides *points* and *costs*. The latter is expressed as *budget left* to the player. The points system was designed to reflect Essence's usage in real endeavors and to reward good practice. In an SE endeavor progress is an intended measure. Essence expresses the progress in fulfilled Checkpoints and reached AlphaStates. Reflecting that, the game rewards fulfilled checkpoints. Any Checkpoint fulfilled for the first time earns 1,000 points. As described in section 3.3 on page 121 Checkpoints may get lost. The pointing system does not reward the loss and subsequent re-fulfillment of Checkpoints because such could be abused easily by players "*gaming the game*." A nice effect of that approach is that a lecturer quickly gets a rough impression about the performance and current position of a player inside the game.

To reward conscious practice, the assessment of a Checkpoint itself gets rewarded, if that one gets checked only after such an assessment is indicated by game's state. If a Checkpoint gets ticked in the *Navigator* and the underlying simulation approves, that it is actually fulfilled, the player gets rewarded with additional 1,000 points. If a checkpoint gets checked in the *Navigator* and the underlying simulation does not approve that it is actually fulfilled, no additional points get assigned. Furthermore, the player can no longer earn the additional points for that one checkpoint once it gets actually fulfilled. Otherwise, it would be possible to abuse the points system by just repeatedly clicking around.

Altogether a player can earn 2,000 points by accomplishing a fulfilled checkpoint and its correct assessment.

As already mentioned in preceding sections the game provides tutor messages if the virtual team got stuck caused by unsatisfied preconditions. To make the difference between proper practice and incorrect one more noticeable, these messages do not get sent immediately but after a configured number of virtual working days resulting in

accrued cost without accomplished progress.

4.7.4.1 Ranking

To provide leaderboards of players in a team, and all the teams of the whole course, the game needs to provide a ranking score that makes results among individual players and teams of players comparable. The ranking score rewards more progress at lesser costs and has to take into account, that one player might be far advanced in the game while others are just starting. The individual ranking score $RS_{P,G}$ of a player P in her game G is defined as

$$RS_{P,G} = Points_{P,G}^2 / Costs_{P,G}.$$

The ranking score emphasizes the points representing progress made and ranks players with same points by their costs accrued to accomplish the points. The ranking score of a team in a course is defined by the mean of the results of all of the team's players.

Current rankings of players inside a team, as well as the team inside the whole course, are presented in the status bar and always visible to the player (cf. figure 4.24). A click at one of the ranking stats reveals corresponding leaderboards.



Figure 4.24: Screenshot of Simulation Game: Ranking Stats Presented In the status bar

4.7.4.2 Leaderboards

The game provides two types of leaderboards: *(in-)team leaderboards* and *course leaderboards*. Both types are shown in figure 4.25. The *team leaderboards* present the rank of a single player inside of his team. The *course leaderboards* present the rank of the whole team inside of the whole course.

Since players might play their game at varying speeds, just one leaderboard per leaderboard type presenting the overall score can't give the desired orientation. A player, who has her game already advanced to game week #15, very likely has made more progress than a player with his game located in game week #2. To make decisions and their results comparable, the game provides a number of leaderboards—one per week and leaderboard type complemented by one all-time leaderboard for both leaderboard types. A game week was chosen as the period to balance between comparable results and a comprehensible leaderboard presentation that does not overwhelm

by too much too fine-grained data. The ranking stats presented at the status bar of the game (cf. figure 4.24) follow the same principle. Instead of showing the all-time ranks they present the rank of the particular game week to enable a meaningful comparison of the quality of decisions made compared to others.

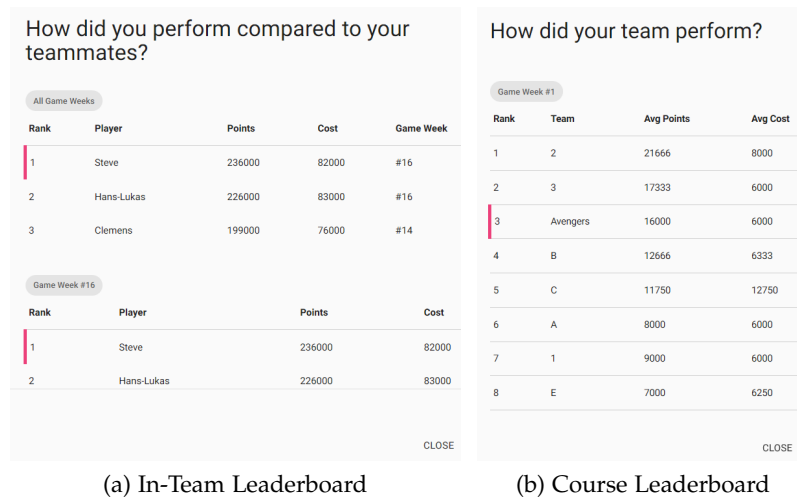


Figure 4.25: Screenshots of Simulation Game: Leaderboards

4.7.5 Teamwork—Collaboration and Competition

Experts in the field of game-based learning demand a “wider use of social interactive learning.”[87]

The game adds the idea of teams to the individual gameplay of every participant. In this setting, each player is exploring the whole game and the integrated learning content on her own but is a member of a team at the same time.

Measuring the performance of the whole team as primary success indicator of the game fosters collaboration and discussion inside the teams trying to perform better than others. This opens new opportunities for interaction. Utilizing the provided leaderboards of team rankings as well as the ranking of individual players gives orientation how optimized chosen paths are in relation to other one’s results. This orientation gives immediate feedback and is the starting point for further interactions inside a team. Since all players try to master the same challenges, a team member performing better at a given time may support other members of the team—having their common goal in mind.

This approach combines collaboration aspects of social interaction inside teams with the competition aspect between teams.

An elementary team chat is provided by the game environment to facilitate communication inside a team (cf. figure 4.17). That’s not to say that this chat should be the only channel of communication.

That would be much too limiting. As observations at performing the case study showed, the chat served only as starting point of active face-to-face communication (cf. chapter 5).

4.7.6 Feedback and Guidance

To support the player mastering the game and hence to learn, the game provides different kinds of feedback and guidance.

While playing the game, a player gets immediate feedback from game's UI. The *Navigator* shows the current state of the endeavor as assessed by the player. Messages from the virtual team inform about the important events at progressing the endeavor and tutor messages, presented as messages from the team too, give guidance as needed (cf. figure 4.31 on page 242). The status bar always provides information about all important measures (cf. figure 4.26 on the facing page). Players are fully responsible for their virtual endeavor but act as a member of a team too. Rankings and leaderboards continually give feedback about player's performance compared to others in the team (cf. 4.7.5 on the previous page) and provide triggers and reasons for social interaction, discussion, and reflection.

After playing the game, the player gets provided with statistics and diagrams inviting for analysis of her chosen solution strategies. Figure 4.27 on the facing page shows three examples of the *Alpha Assessment Graph* presented to the player. The three diagrams show different player results. The red areas of the graph show, how the players assessed the Alphas of their endeavors. The gray areas visualize the actual state of the endeavor as calculated by the simulation. A graph with two completely congruent areas represents a perfect endeavor assessment (c). We can see that the first player (a) assessed all of his Alphas of the *endeavor* area of concern (*Team, Work, Way of Working*) as much too progressed but the *Alpha Stakeholders* too reluctant. This player most likely ignored messages of his virtual team, indicating that some of the states once achieved got lost, which resulted in a decreased team performance. Partially this applies to the results of player (b) too. His *Team Alpha's* progress also was assessed too high. He assessed the other Alphas of the *endeavor* area very well but failed at assessing the Alphas of the *solution* area of concern (*Requirements, Software System*). These three results show that players acted differently and were focused on different areas of the endeavor.

Figure 4.28 on page 242 shows the performance of the virtual team in detail. All players started to progress the Alphas of the *endeavor* area of concern almost at the same time. While the first player (a) stopped activities progressing those Alphas, the other two players progressed them further. Player (b) stopped at 90% of team performance while the player (c) progressed the Alphas to team's full performance—and kept them in those states. Player (b) stopped one or



Figure 4.26: Screenshot of Simulation Game: Status Bar Details

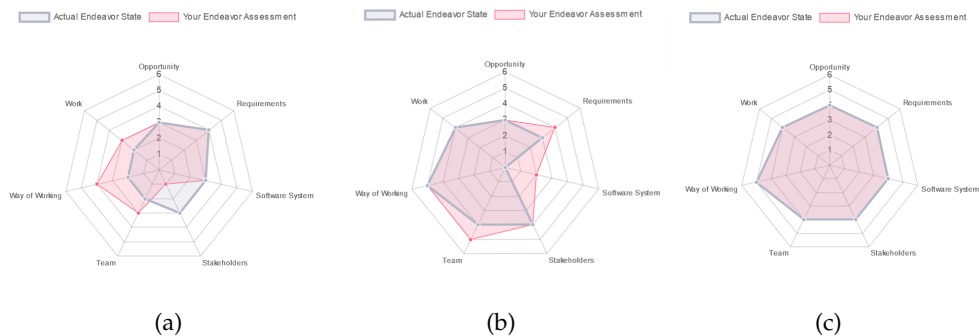


Figure 4.27: Screenshots of Simulation Game Results: Alpha Assessment Comparison

more of the activities (*coordinate activity, support the team, track progress*) and team's performance decreased.

Figure 4.29 on the following page provides another perspective on the correctness of player's checkpoint assessments. The graph combines the number of checkpoints assessed correctly (green area/line), the number of checkpoints ticked without cause (red area/line) and the number of checkpoints that were not ticked although this had been indicated. We can see that player (a) stopped quite early to assess his checkpoints correctly and did not tick checkpoints that had been fulfilled. Player (b) performed better at checkpoint assessing. We see a considerably lower number of wrongly non-ticked checkpoints and from time to time some checkpoints ticked without cause. Player (c) did it almost perfectly. Only a few checkpoints were ticked too early over the whole time of playing.

Figure 4.29 on the next page show two other diagrams provided at the end of the game. Diagram (a) presents costs cumulating over time and diagram (b) shows how player's score developed over time.

Looking at all these graphs, players are invited to recognize strengths and weaknesses of their assessments and their resulting decisions. They are enabled to analyze at which time failures caused wrong decisions.

By comparing and discussing these results at debriefing time, students benefit from results and experiences of other players.

4.7.7 Where Are the Levels and Game Badges?

Games usually provide levels with raising the degree of difficulty and provide badges to players rewarding and visualizing achievements in

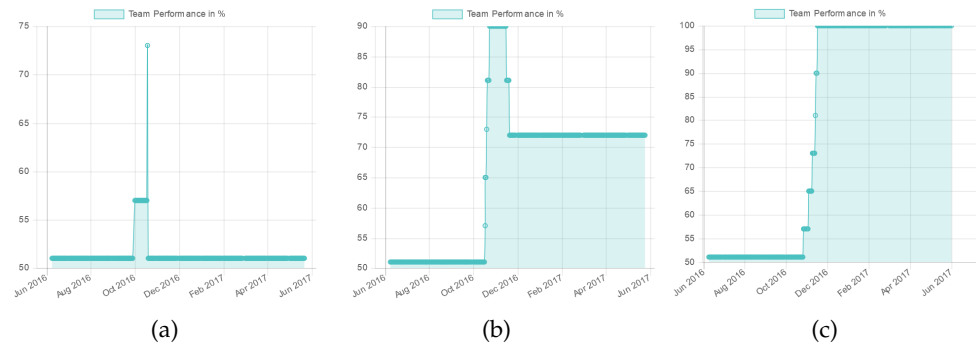


Figure 4.28: Screenshots of Simulation Game Results: Team Performance Comparison

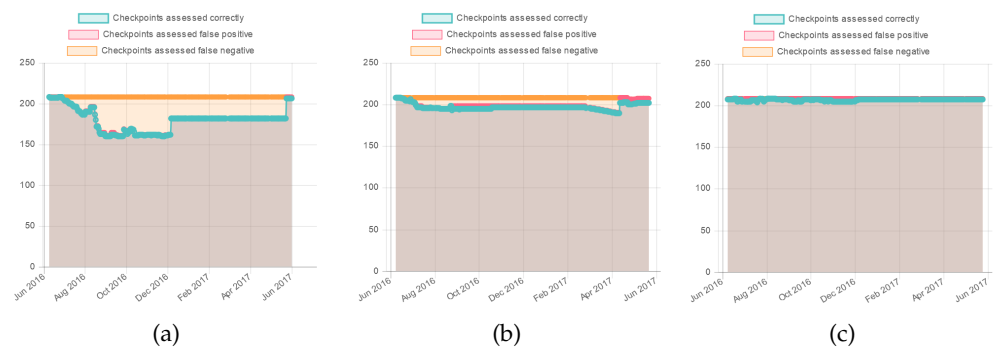


Figure 4.29: Screenshots of Simulation Game Results: Checkpoint Correctness

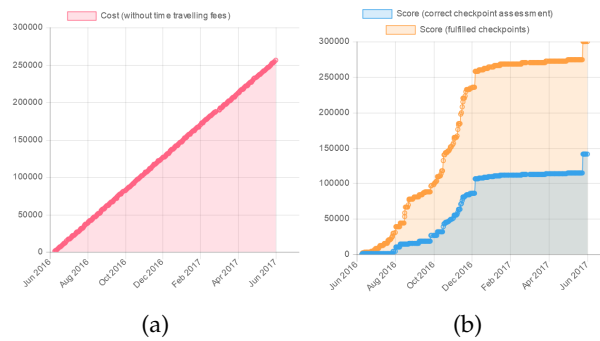


Figure 4.30: Screenshots of Simulation Game Results: Cost and Score



Figure 4.31: Screenshots of Simulation Game: Confirmation Message and Tutor Message

the game. The *Simulation Game* provides nothing of that kind—by intention.

Instead of artificially dividing the (game) flow of progressing an SE endeavor, the game makes use of the natural structure and “*rewarding system*” of the underlying domain. Essence provides Alphas with their AlphaStates and their Checkpoints. Each time the player ticks off a Checkpoint (and that action was indicated) she gets rewarded by points and the *Navigator* visualizes each progress stated, e.g. via the balance indicator (cf. section 4.6.2.2). Seen from this perspective a player earns the same “*badges*” in the game as he does as a software engineer at driving a real SE endeavor.

Reaching an Alpha State of an Alpha might be seen as completing a level-like stage of the game. The difficulty of the game increases naturally, driven by the inherent characteristics of the domain. The number of activities that can be (reasonably) assigned to the virtual team, grows while the game is progressing. By assigning multiple activities to the virtual team at once, the number and variety of information flowing in are growing, making the proper processing and analysis of the current state of the endeavor more challenging. The decreasing performance of the virtual team, caused by the insufficient attention paid to the Alphas of the *Endeavor* area of concern, might add new challenges too, etc..

With this characteristics, this approach provides an *intrinsic, tightly integrated* game utilizing a *blended paradigm* (cf. section 4.7.11 on page 245). The author is aware that this approach might sound too puristic and optimistic to some proponents of the game-based learning community, but—as the results of the case study show—it worked out well at its conducting (cf. chapter 5).

4.7.8 Lecturers Point of View

In a digital game-based and generally in a constructivist learning environment the lecturer acts as a facilitator and enabler. To get an overview of the game-playing performance of all participants in a course and to give support as needed, the lecturer is supported by the game environment.

The game environment provides elementary course management capabilities to organize students into teams and to provide a game and underlying simulation to each of the players. Furthermore, a lecturer might impersonate each of the players, getting the same perspective as the respective player impersonated. Having a look at the course leaderboard, the lecturer quickly gets an impression of the results of all the teams so far, as the points and cost are reflecting essential measures of the game and are easy to interpret (cf. section 4.7.4 on page 237).

4.7.9 *Debriefing*

Debriefing activities play a vital role at game-based learning approaches.[301] They offer the chance to discuss results as well as to compare individual experiences and different solution strategies of the players. Such activities complete the learning experience and invite to reflect on experiences made. Simplifications and inaccuracies of the game might get discussed and qualified.

Debriefing questions used in the evaluation experiment included the following:

- What activities of your virtual team did surprise you the most?
- Did you get stuck somewhere in the endeavor? Where? How did you get back to a healthy and progressing endeavor?
- When did your team notify that it would not make any progress? How much time did it need to recognize that? Do you think that is realistic?
- Some of you got stuck at different positions in the game because of preconditions missed to start the work on an activity. How would you manage this in a real project?
- How much time did your team spend to achieve the results in the game? Do you think that is realistic?
- How did the number of team members evolve? Do you think that it is always that way?
- How many team members did your team consist of? What was the cost of your team per day? Do you think that is realistic?
- Do you think your team members did all share the same experiences and can accomplish all the tasks with same quality and performance? Why (not)?
- Did you do a lot of time traveling in the game? What did you do in a different way once back in the past? How would you act in a real project where time traveling is not an option?
- What kind or category of software process did your team use in the game? Why?
- What practices did your team use to accomplish {requirements elicitation | a tested system | ... }?

Students participating in the evaluation experiment highly appreciated debriefing and attributed learning to it (cf. chapter 5).

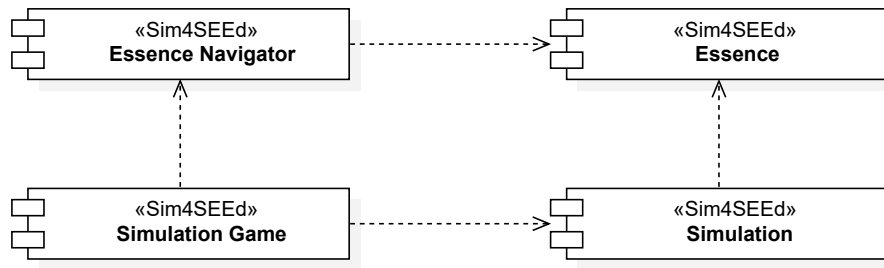


Figure 4.32: Components Architecture

4.7.10 Architecture

The *Simulation Game* system employs a set of components implemented for this approach and is based on web technologies to a great extent. Figure 4.32 shows a high-level overview of the components involved and their dependencies.

The game system provides an elementary course management system enabling lecturers to administrate courses with students and teams. Each student in a course is associated with a team and owns one game instance per course. Each game instance owns a simulation driving the game. Simulations run in their own process.

The game system was implemented using open source software. To provide leaderboards and inter-process authentication, a key-value-store²² is employed. The *Navigator* stores method definitions (at level 1) in a document based database²³ and all data needed to run and track endeavors in a relational database²⁴. All components are implemented utilizing the Ruby programming language²⁵ supported by a number of great Ruby gems. Game UI, as well as the Navigator UI, make heavy use of JavaScript libraries²⁶. Their backend is implemented using the Ruby on Rails web development framework²⁷. To enable immediate feedback inside the game, a push service²⁸, providing WebSocket connections and fallback solutions, is utilized.

4.7.11 Classification of Taken Approaches

The *Simulation Game* is, regarding the classification of Ritterfeld and Weber[229], representing a *blending paradigm* cf. figure 2.6 where entertainment and learning are not sequenced but integrated. The entertaining or rewarding elements draw exclusively on the inherent domain (cf. section 4.7.7).

²² <http://redis.io/>

²³ <https://www.mongodb.com/community>

²⁴ <https://www.postgresql.org/>

²⁵ <https://www.ruby-lang.org>

²⁶ e..g. React (<https://facebook.github.io/react/>) and Redux (<http://redux.js.org/>)

²⁷ <http://rubyonrails.org/>

²⁸ <https://faye.jcoglan.com/>

Game Classifications

With regards to the flexible categorization approach supposed by Breuer and Bente[40, 41] the *Essence Kernel Puzzler* and the *Simulation Game* would be classified with the labels summarized in table 4.7.

By utilizing Prensky's[218] categorizations, the *Simulation Game*

- is an *intrinsic* (not an extrinsic) game, since the learning content is an integral part of the gameplay,
- is a *tightly linked* (not a loosely linked) game, since the learning content is built into the game and knowing it is vital to succeeding in the game,
- is a *reflective* (not an action) game, since it allows, or requires, a high degree of reflection by providing as much time as needed to (re-)think about decisions to make,
- is an *asynchronous* (*turn-based* not a synchronous real-time) game, since the player, after having made her decisions, taken all the time needed, has to resume the game,
- is neither a single-player, two-player nor multiplayer game, it combines a *single-player* game experience with a *team-based collaboration* and *competition* approach,
- is a *persistent-state* (not a session-based) game, since the state of the game is persisted and players may resume the game later on.

4.7.12 Future Work

Results of running the case study are presented in the next chapter 5.

After playing the game, a number of students asked for some tutorial or help system added to the game. At running the case study, the author was available on site and questions or technical problems were answered quickly. To support a wide usage of the *Simulation Game*, some introductory tutorial will be part of future work.

The current course management system and system configuration served their purpose in the case study but require remarkable knowledge about the system at the moment. Hence they are not intuitively usable by novices. The interface connecting game and simulation environments needs some rigorous scalability testing to ensure support in bigger environments.

4.8 RELATED WORK

The *Integrated Approach* presented in this thesis builds upon existing approaches (cf. section 2.5 on page 68) and their findings. This sec-

Label / Tag / Category	Essence Kernel Puzzler	Essence Simulation Game
Platform	Web-browser	Web-browser
Subject Matter	SEMAT Essence Kernel elements, their definitions and their relations	SEMAT Essence Kernel, elements, their interdependencies, Plan-Do-Check-Adapt cycle of driving an SE endeavor
Learning Goals	Essence vocab and basic concepts of Essence elements and their relationships	application of Essence concepts, esp. the PDCA cycle (cf. section 2.7)
Learning Principles	rote memorization, exploration of dependencies	exploration, situated learning, guided/scaffolded discovery learning, PBL, anchored instruction
Target Audience	SE students, SE practitioners	SE students, SE practitioners
Interaction Mode(s)	single player	team enhanced single player
Application Area	primarily academic SE education, training	primarily academic SE education, training
Controls/Interface	mouse	mouse & keyboard
Common Gaming Lables	puzzle(r)	simulation

Table 4.7: Games Classification By Labels[40, 41]

tion compares it with existing approaches of DGBL in SE education that are addressing software processes, SE methods, and their management. Similarities, as well as differences from existing approaches, get summarized.

Like in other existing approaches, a player in the *Simulation Game* (Phase 2 of the *Integrated Approach*) takes a role as project manager, or rather a *coach*²⁹, to control an SE endeavor by directing a virtual team through a virtual software project.

The *Integrated Approach* is oriented towards supporting different SE methods like *SimSE*, *SESAM* and *Nassal's Project Management Game* (PMG). In the conducted case study the SEMAT Essence Kernel was utilized. The approach already allows the use of other Essence methods conceptionally but does not yet support it technically. Different from existing approaches the *Integrated Approach* is the first one utilizing an industry standard for standardized method description[188]. This opens the opportunity to build simulation models for arbitrary SE methods following the Essence concepts. The *Integrated Approach* is built on SEMAT Essence. With its kernel approach, Essence is providing students with a highly transferable thinking framework (cf. 2.7 on page 95) supporting a demanded orientation toward the use of flexible SE practices instead of monolithic software processes (cf. 2.6.1 on page 88).

The chosen simulation paradigm is different from existing approaches. With the *Discrete Event System Specification* (DEVS), a specific DES formalism with sound theoretical background was chosen. Instead of implementing a DES model from scratch, making it harder for others to utilize and extend the approach, the formalized, well described, and widely accepted DEVS provides a convenient and flexible starting point. Further advantages already got discussed in chapter 3.

The approach taken does neither provide a specific DSL like *SESAM* or *Nassal's PMG* nor a rule editor like *SimSE* but follows an own modeling approach to building simulation models. Following Squire's notion of games being *designed experiences*[263], the process of modeling does not require to design and build formulas or to start from scratch to build a simulation model. The modeling process is much more similar to writing a screenplay with a timeline of events, represented by Essence Checkpoints and messages of the virtual team—actually multiple timelines since the defined interdependencies of the Checkpoints and the Alpha States allow for multiple ways through an SE endeavor. This approach needs less training effort than existing ap-

²⁹ since emerging agile and lean approaches often do not include a classical project manager role[8, 141]

proaches for building and configuring the simulation model and requires only knowledge of Essence concepts. The *Integrated Approach* strives for using synergies, thinking about the simulation model will result in a better understanding and utilizing of Essence in real world projects.

Like *Nassal's PMG* the *Integrated Approach* makes use of real-world tools inside of the *Simulation Game*, but due to a lack of existing tools to support driving SE endeavors with Essence, it brings its own tool built for that purpose: the Essence Navigator. The *Navigator* gets utilized inside the *Simulation Game* in the same way as in the following real project work hence it provides an already familiar environment at real project's start and lowering the cognitive load of students in the real project work. Differing from *Nassal's PMG* the utilized tools is compatible with plan-driven as well as agile and lean principles and SE methods.

Like *Nassal's PMG* the environment of the *Integrated Approach* and the *Simulation Game* was componentized, enabling partial reuse, e.g. by replacing the game UI with another one.

Different from existing approaches this one is not focused on workload optimization of single project members, since such a focus is less compatible with emerging agile or lean concepts and occupying learners' attention to a high degree, hence interfering the recognition of more essential concepts and relationships while driving an SE endeavor.

This approach aims at preparing and providing direct support for students in course or capstone project work that is demanded by all curriculum guidelines (cf. section 2.3 on page 38). If SE students should acquire an SE mindset, appreciating the guidance and support of utilized methods and practices they should be provided with experiences facilitating such perceptions. The *Integrated Approach* aims at facilitating such experiences by providing scaffolding, a common anchor, and lowering the cognitive load of students in course projects.

The *Integrated Approach* takes a diametrically opposite position of the *MO-SEProcess* (cf. section 2.5 on page 68). Instead of exporting collaborative activities into artificial 3D worlds, the approach aims at triggering and providing occasions for social interaction, preferably in the real world and in direct communication face-to-face, since this kind of communication is much richer and omits any technical limitations.

Drawing on *social constructivism* theories (cf. section on page 19) this approach promotes social interaction and collaboration. With that, this approach shows parallels to the one of Caulfield's *Simsoft*

but is different in the fact that everyone has to cognitively accomplish her own virtual endeavor. Different from *Simsoft's* approach, this one adds competition by providing team as well as course rankings and leaderboards. Beside challenging competition these components provide feedback and orientation at assessing one's performance in relation to other one's, hence provide additional occasions for social interaction and reflection. None of the existent approaches utilized collaboration and competition in that way.

The *Integrated Approach* takes a diametrically opposite position of the explicitly non-guided *SimjavaSP* by providing carefully designed scaffolding. Besides providing a constructivist learning environment (cf. section 4.1.2), the introduced approach considers Vygotsky's *Zone of Proximal Development (ZPD)*, Sweller's *Cognitive Load Theory (CLT)* and Reigeluth's *Elaboration Theory* (cf. section 2.2 on page 19) to provide an effective learning environment. By integrating the description of SE method elements into the gameplay via the *Essence Navigator*, this approach follows an approach somewhat similar to *Nassal's PMG*, but by utilizing SEMAT Essence, the elements are not just documented but *actionable* and *dynamically supporting* students at driving an SE endeavor (cf. section 2.7.2.1 on page 105).

Acknowledging existing findings that games alone are not sufficient learning vehicles and hence have to be complemented by other educational approaches and to be integrated into the learning context to address intended learning objectives, the *Integrated Approach* was designed. It combines a number of non-game activities with DGBL in a well-matched manner to provide an anchor for further activities (cf. section 4.3). This is the first approach explicitly preparing students for and supporting students at their course/capstone project work that is demanded by curriculum guidelines.

APPLICATION AND EVALUATION

To evaluate the *Integrated Approach*, its concepts, games, and tools, it was deployed in a case study. This chapter describes the settings of the case study, its evaluation considerations, and summarizes the results of its evaluation.

5.1 CASE STUDY SETTINGS

This section describes the setup of the case study used to evaluate the chosen approach. Two groups of participants took part in the case study. Both consisted of students of the *University of Applied Sciences Stralsund, Germany*, and get described next.

5.1.1 Characteristics of Participating Groups

Two participating groups were employed in the summer semester of 2016. They provided the observations and results reported in the following sections.

The first group (*Group 1*) consisted of 12 students (11 male, 1 female) taking the course “*SMIB4500—Project Seminar Software Engineering*” embedded in the 4th semester of the undergraduate curriculum of “*SMIB—Applied Computer Science, Software Development and Media Informatics*.” Within the scope of the course, students got introduced to SEMAT Essence for the first time. Traditionally in the course, students exercise a software project from start to end, following a given software process based on the *OpenUP* [101, 226] and *AgileUP* [57, 79] and heavily customized to the organizational needs of the course. Customized that way, this software process should not be utilized in a real SE endeavor. Students got introduced to SEMAT Essence and in particular the Essence Kernel to get provided with transferable concepts and a thinking framework being of use in any future SE endeavor.

The second group (*Group 2*) consisted of 7 volunteers, visiting the 6th semester of the undergraduate program “*SMIB—Applied Computer Science, Software Development and Media Informatics*” and interested in learning something about SE in general and Essence in special. This group differs to some extent from *Group 1*. One year ago they got shortly introduced to SEMAT Essence. Generally, students of *SMIB* in 6th semester already had their practical semester where

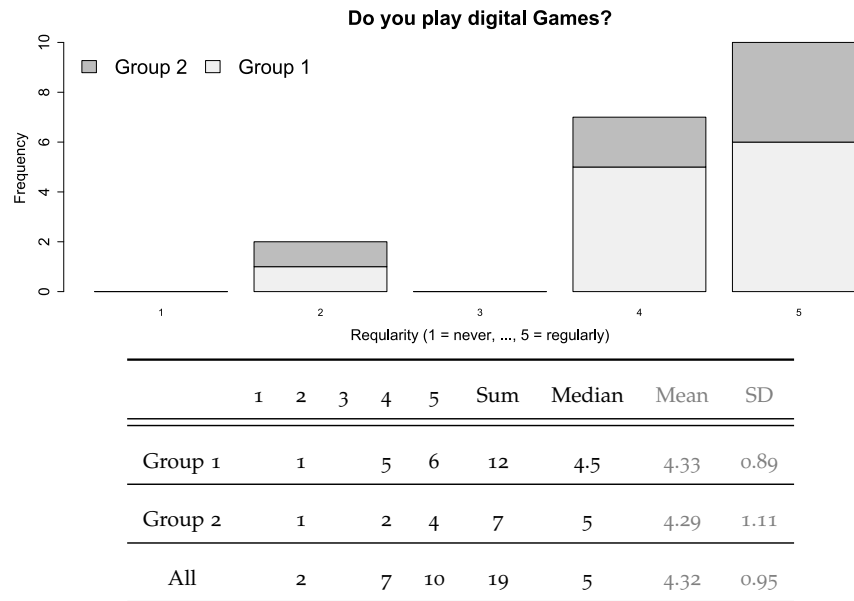


Figure 5.1: Q01: Do you Play Digital Games?

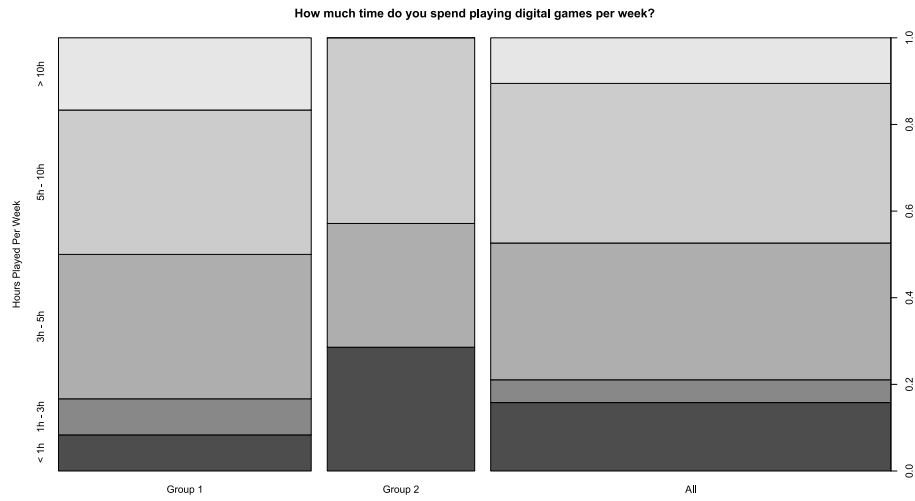
they gain experience in real software industry. In their 6th semester, they are processing a bigger software project within the scope of a course called “SMIB7220—Software Project Organization”. Within that course, they got in touch with Essence again and were already used to work with Essence Kernel Cards to some extent. Students of *Group 2* were only invited to try the *Essence Kernel Puzzler* and to play the *Simulation Game*.

Figure 5.1 illustrates that students of both groups are enjoying digital games. 53% (n=10) play regularly and further 37% (n=7) to a lesser extent.

The spine graph in figure 5.2 on the next page illustrates the time spent playing digital games. Compared to a German survey of gamers taken in 2011, the numbers show that the participants play more hours than the average gamers between 14 and 69 in 2011[59]. 47.3% of the participants play more than 5 hours per week (compared to 28% in a German survey 2011[59]).

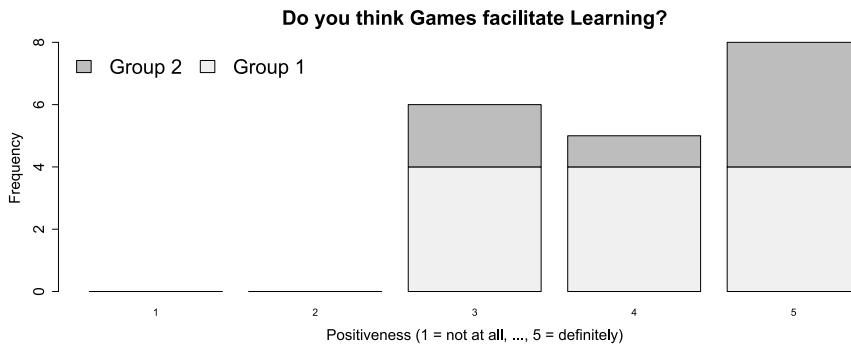
68% of the participants connect learning with games. The remaining 32% of participants chose the neutral answer item. Hence none of the participants disagrees with the statement that “games provide learning.”

58% (n=11) of all participants stated to have already collected practical experience in software projects outside of the curriculum. Obviously, the share of students with that kind of experience is much bigger in *Group 2*, since students in the 6th semester regularly went through their practical semester to gain experience in real software



	< 1h		1h - 3h		3h - 5h		5h - 10h		> 10h		Sum
Group 1	1	(8.3%)	1	(8.3%)	4	(33.3%)	4	(33.3%)	2	(16.7%)	12
Group 2	2	(28.6%)			2	(28.6%)	3	(42.9%)			7
All	3	(15.8%)	1	(5.3%)	6	(31.6%)	7	(36.8%)	2	(10.5%)	19

Figure 5.2: Q03: How much time do you spend playing games per week?



	1	2	3	4	5	Sum	Median	Mean	SD
Group 1			4	4	4	12	4	4.00	0.85
Group 2			2	1	4	7	5	4.29	0.95
All			6	5	8	19	4	4.11	0.88

Figure 5.3: Q06: Do you think games provide learning?

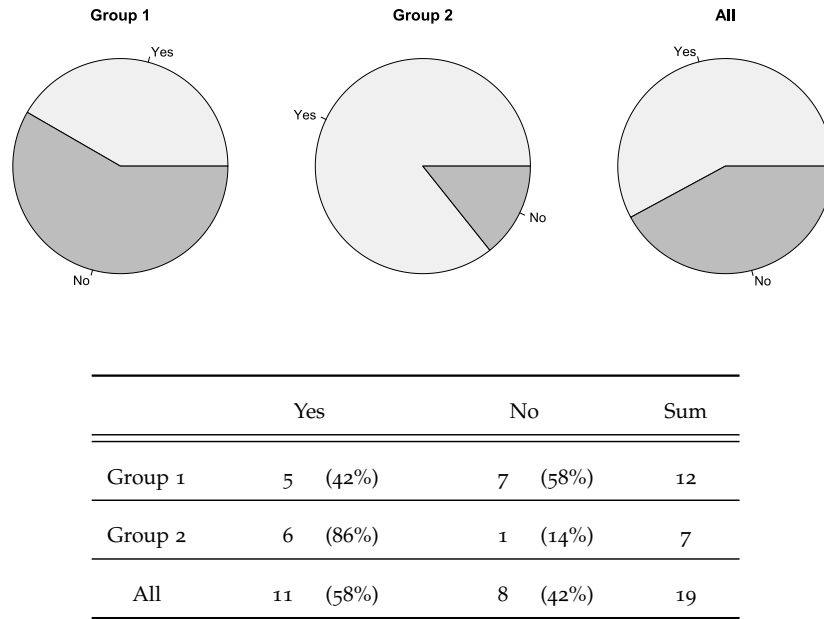


Figure 5.4: Q07: Did you already work on software projects outside of your curriculum?

industry. In *Group 1* 58% ($n=7$) of the participants had no experiences at software development outside of their curriculum.

5.1.2 Procedure of Group 1

Within the regular course students of *Group 1* were introduced into SEMAT Essence by an educational case study[244], presenting a small team of software developers at kickstart of a software project. This case study is provided by SEMAT.

Students were invited to play the *Essence Kernel Puzzler* in preparation of the *simulation game*. At the next course meeting, a few days later, two course units of 90 minutes were used for playing the *simulation game* and a debriefing session. The students were already organized into 4 teams for their project work. These project teams were used for the game too. The author provided an introduction into the gameplay, provided technical support, answered questions regarding the game, and moderated the debriefing session (cf. 4.7.9 on page 244). Finally, students filled in the questionnaire.

After this session, students used the *Essence Navigator* in their project work. The tool supported project team meetings and helped teams to keep on track. The Essence Navigator provided an enacted Essence Kernel. Since the process to follow was not modeled as Essence practices forming an Essence method, students had to do some mental mapping between Essence concepts, e.g. Activity Spaces and activities required by the aforementioned given software process to follow.

In an optional exercise, the teams were asked to map explicitly

- the assigned activities of the given process to Activity Spaces of the Essence Kernel and
- the points in time, when all the Alpha States got achieved, to phases and concrete iterations of the given software process.

The Alpha States that were not part of the assigned project work had to be marked. At each mapping the students were asked to provide a short explanation of the specific mapping done.

5.1.3 Procedure of Group 2

Group 2 consisted of volunteers following an open invitation. Students were invited by the author to take part in a simulation game session and (optionally) to take the *Essence Kernel Puzzler* in preparation of the *simulation game*. Since students already had contact with SEMAT Essence, no further preparation was provided. The group was partitioned into three teams. In one session, lasting two hours, the *simulation game* was played followed by the debriefing session. The author provided an introduction into the gameplay, provided technical support and answered questions regarding the game. Finally, the questionnaire was filled in by the students.

5.1.4 Why Is There No Comparative Experiment?

Having only access to quite small groups of students for experiments/-case study, it was chosen to provide all of them with the chance to learn from new concepts and approaches and to collect as much feedback on the tools and concepts developed as possible. With just 19 participants a division into treatment and control groups would have made the results reported in section 5.2 on page 258 less informative and significant.

Furthermore, SEMAT Essence is a new standard with growing—but at time of conducting—limited learning material. With no prepared rich and ready to use learning material, rated competitive to the provided approach, any attempt to produce that material alongside with limited resources and less persuasion was judged to be in danger of getting exposed to a biased performance.

While appreciating comparative studies for their expressiveness regarding ensuring efficiency, a comparative experiment was not performed for reasons given above in the first place—leaving the door open to such an endeavor once richer learning material and bigger subject groups get available.

5.1.5 Why Are There No Pre-Test/Post-Tests?

The skills and competencies striven for in this approach and facilitated by the games provided are to orientate inside an SE endeavor. This includes utilizing SE concepts and tools provided to act in an SE endeavor in a goal-oriented and structured way, e.g. by holistically assessing the current state of an SE endeavor and to determine next reasonable steps to take (cf. figure 2.16). Such skills and the corresponding needed attitudes are not, or only to a small extent, measurable in a test.

Rote memorization is not a primary concept of Essence, which provides the foundation of the *Integrated Approach*. Instead, it is providing rich as well as compact documentation, e.g. in the form of tangible cards to carry with you. There is no need to rote memorization of all states and their checkpoints. Essence promotes learning by doing and enables to learn and deepen knowledge on the fly—once basic concepts are understood.

The *Essence Kernel Puzzler*, as well as the *Simulation Game*, provide some *embedded assessment*[255], in-game measurements allowing to draw inferences from the gameplay about the learning progress to some extent. This approach avoids disadvantages associated with examinations of learning progress separated from the gameplay reported by existing studies of DGBL[255].

5.1.6 Questionnaire Design

The questionnaire was designed to survey different aspects of the approaches taken.

Of particular interest for the evaluation are

1. background information, including participant's attitudes towards digital games, learning through games and favored learning approaches,
2. participants' assessment of the simulation game, including the perceived fun, duration, difficulty, and favorite as well as potentially confusing aspects,
3. the perceived learning effect and which aspects affected learning the most,
4. participants' assessment of the *Essence Kernel Puzzler*,
5. participants' assessment of the *Essence Navigator*, and
6. participants' assessment of SEMAT Essence's utility in future SE challenges.

The answer types include dichotomic choices, 5-point Likert-type scales and other rating scales resulting in ordinal scaled data, as well as free text answers to collect remarks of participants.

Results of the questionnaire are presented in detail in the following sections.

5.1.7 Statistical Hypothesis Testing

To establish statistical significance answers undergone hypothesis testing where appropriate. Many of the questions answered by the participants utilize a 5-point Likert-type scale or another rating scale resulting in *ordinal scaled* data. Ordinal scales allow for ranked orders resulting in sortable data but not for statements about the distances between their items. They are not necessarily equidistant hence not adequate for calculations, e.g. calculations of mean and standard deviation. But publications in the field provide such data. Having noted this, values for mean and standard deviation are provided for reasons of comparability. To be in contrast, such values are printed in a lighter gray.

Having only a small set of observations ($n < 20$) without any knowledge about the particular distribution of the data, only a *non-parametric* (“*distribution free*”) statistical test, able to handle small sets of observations, is qualified for this study. The *Sign-test* fulfilling these criteria was chosen. It is a non-parametric binomial test about the median η of a population.

5.1.8 Heuristics

To establish the consideration of relevant dimensions, the characteristics of the *Integrated Approach* and the learning games provided were mapped to quality criteria of learning games proposed by Dondi and Moretti[74] as result of two European research projects involving organizations from different European countries, backgrounds, and expertise. This mapping is presented in the appendix C on page 361. The mapping ensures that all relevant dimensions of learning games were considered and the demanded quality criteria are provided.

Two additional mappings presented in the appendix C on page 361 compare the *Integrated Approach* and its components with

- recommendations based on the synthesis of a comprehensive SLR[126] (cf. section C.3 on page 366), as well as
- SE education requirements based on conclusions of Boehm[27] reviewing SE in the 20th and 21st century (cf. section C.2 on page 366)

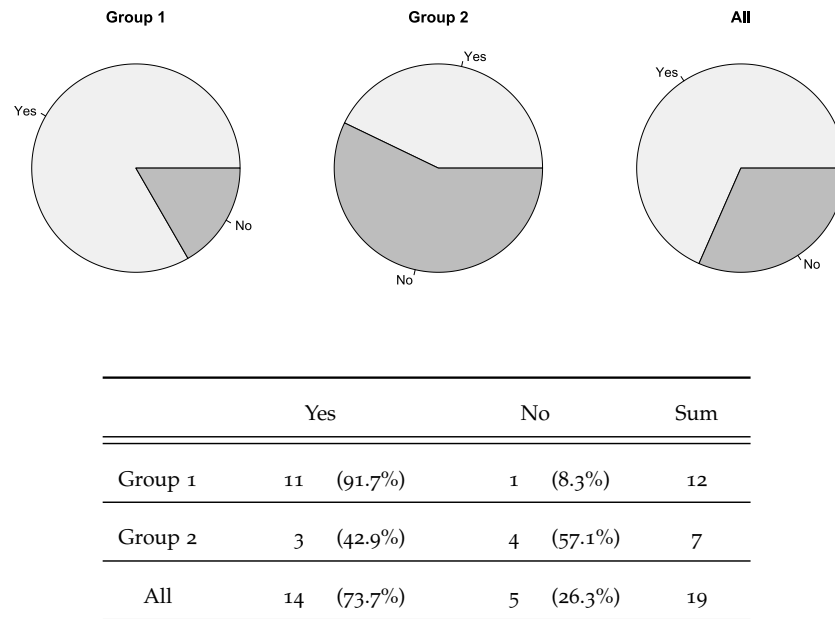


Figure 5.5: Q31: Did you use the Puzzler to be prepared for the Simulation Game?

to enable the assessment of their contribution to the stated learning objectives (cf. section 4.1.3.1 on page 186) and modern SE education in general.

5.2 RESULTS

To evaluate the approach taken, participating students were asked to answer a questionnaire (cf. chapter E on page 373). Results of its analysis are presented in the following section. These results are complemented by measures taken while students were playing the games provided.

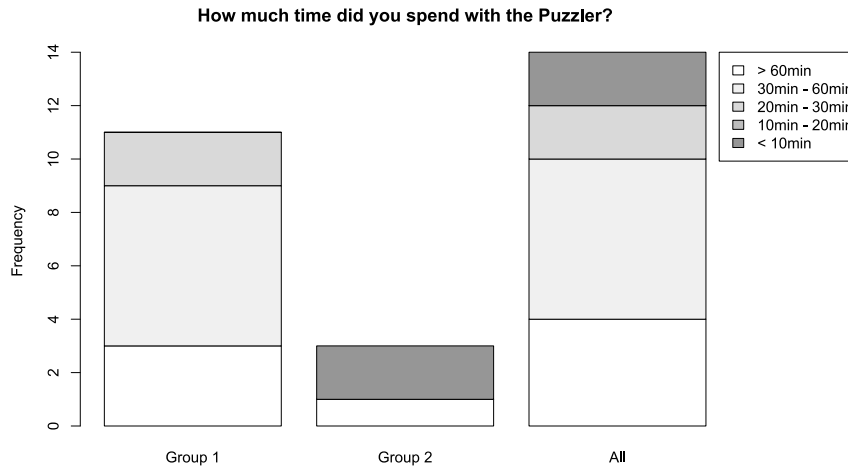
5.2.1 Essence Kernel Puzzler

This section presents results of the questionnaire with regards to the Essence Kernel Puzzler¹. In-game measurements already got presented in section 4.5.5 on page 209.

83% of Group 1 (n=10) and 43% of Group 2 (n=3) stated that they used the Puzzler to get prepared for the *Simulation Game* (cf. figure 5.5).

All Students of Group 1 stated that they spent more than 20 minutes playing the *Puzzler*, 55% (n=6) stated they spent between 30 and 60 minutes playing the *Puzzler* and 27% (n=3) spent more than 60

¹ These questions were marked as optional since the use of the *Puzzler* was presented as optional. It seems that one participant, who took the *Puzzler*, did not answer all the questions, resulting in varying sums of respondents' answers (13 vs. 14).



	< 10min	10min - 20min	20min - 30min	30min - 60min	> 60min	Sum
Group 1			2 (18.2%)	6 (54.5%)	3 (27.3%)	11
Group 2	2 (66.7%)				1 (33.3%)	3
All	2 (14.3%)		2 (14.3%)	6 (42.9%)	4 (28.6%)	14

Figure 5.6: Q32: How much time did you spend with the Puzzler?

minutes with the *Puzzler* (cf. figure 5.6). Students of Group 2, who already had more prior contact with *Essence*, spent remarkably less time with the *Puzzler*. 67% (n=2) stated that they spent less than 10 minutes with the *Puzzler* while 33% (n=1) stated to have spent more than 60 minutes with it (cf. figure 5.6).

The results indicate that the *Essence Kernel Puzzler* helps to get familiar with *Essence* concepts and vocab (Sign-test, $\alpha = 0.05$, cf. figure 5.7 on the following page and table D.1 on page 372).

77% (n=10) of the players assessed the difficulty of the *Puzzler* as “just right,” 23% (n=3) found it “slightly too easy” (cf. figure 5.8 on the following page).

Players would use the *Puzzler* again to reinforce *Essence Kernel* vocab (Sign-test, $\alpha = 0.05$, cf. figure 5.9 on page 261 and table D.1 on page 372).

77% (n=10) of the respondents stated that the *Puzzler* prepared them well for the *Simulation Game* (cf. figure 5.10 on page 261).

92% of responding students (n= 12) would recommend the *Puzzler* to a friend or fellow student (cf. figure 5.11 on page 262). Asked for their reasons, why they would recommend the *Puzzler*, students stated they found it to be a neat tool to learn the vocab fast and to get familiar with first relationships between elements. They emphasized that while it is a good tool to learn the vocab, it cannot provide the experience to apply the concepts close to reality.

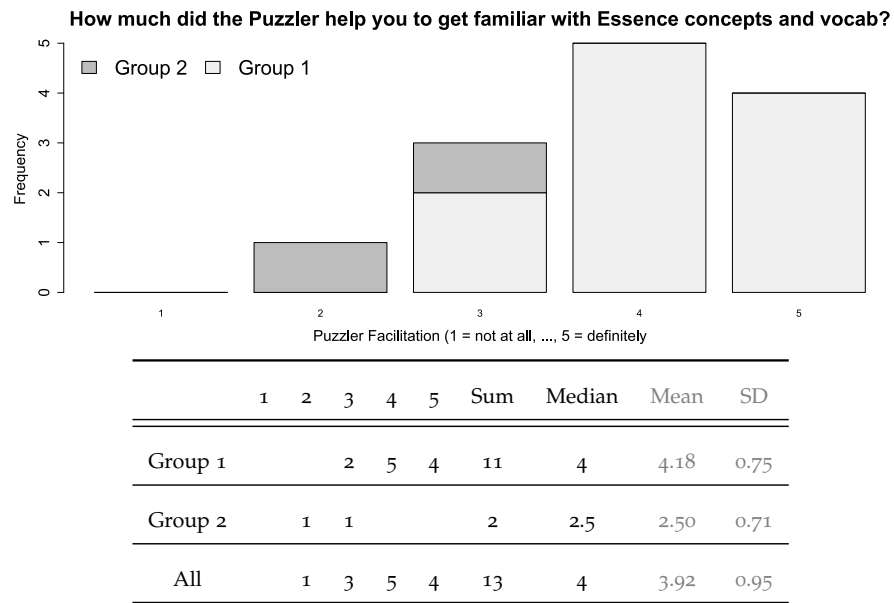


Figure 5.7: Q33: How much did the Puzzler help you to get familiar with Essence concepts and vocab?

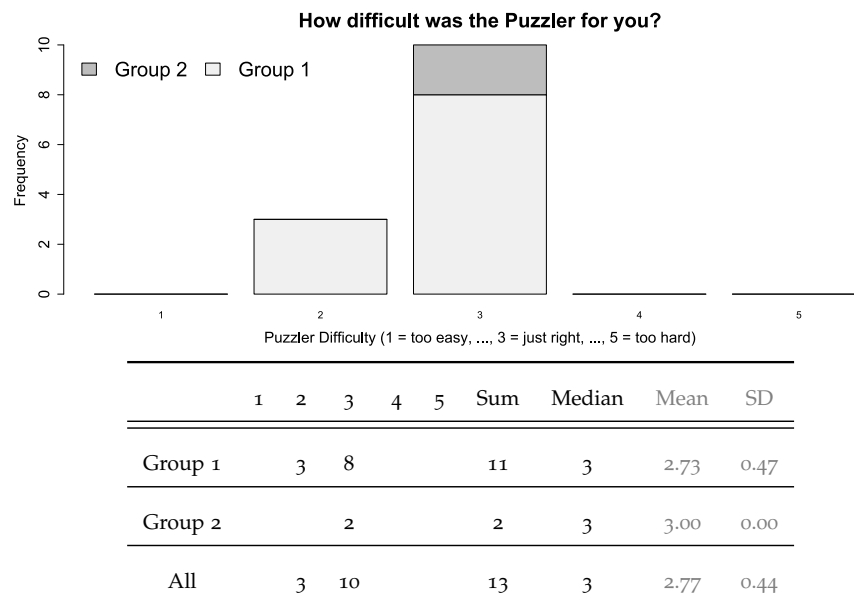


Figure 5.8: Q34: How difficult was the Puzzler for you?

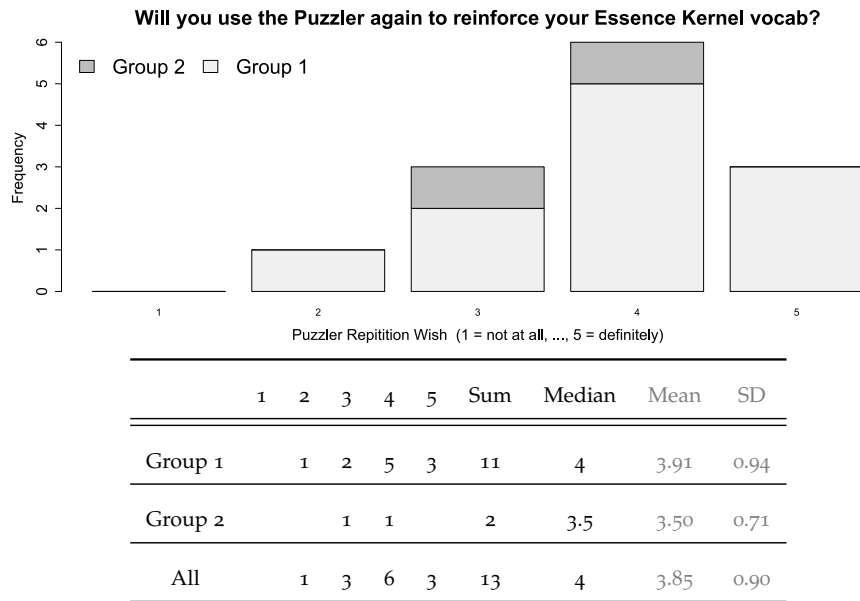


Figure 5.9: Q35: Will you use the Puzzler again to reinforce your Essence Kernel vocab?

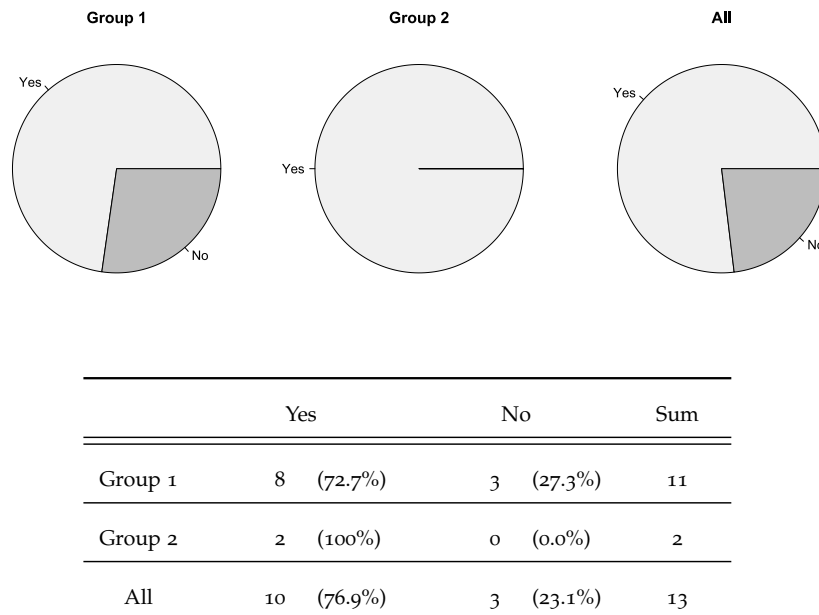


Figure 5.10: Q36: Did the Puzzler prepare you well for the Simulation Game?

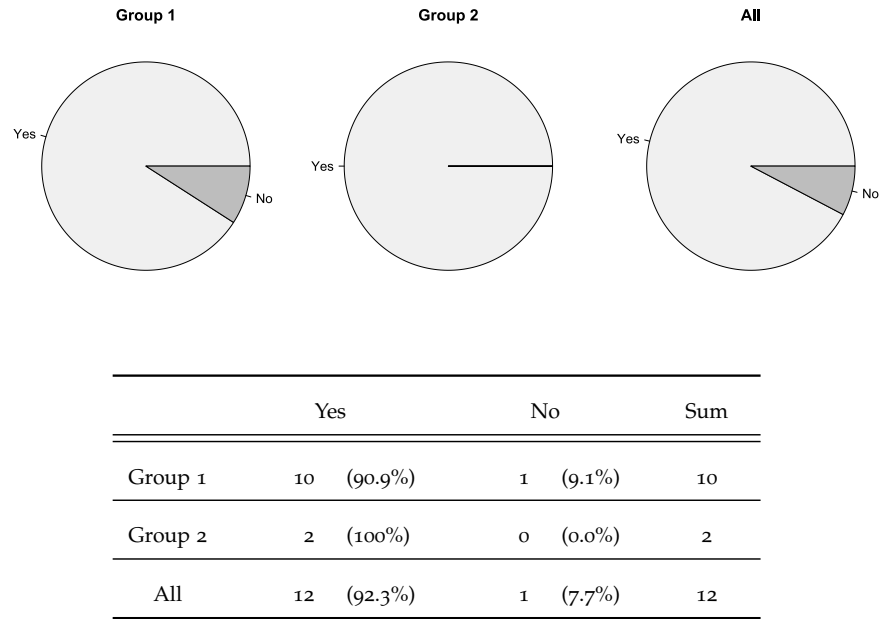


Figure 5.11: Q37: Would you recommend the Puzzler to a friend/fellow student?

5.2.2 Simulation Game

This section presents results of the case study where the *Simulation Game* was deployed. Measured results get presented first, followed by the results of the questionnaire conducted after debriefing activities, which followed the gameplay.

5.2.2.1 Remarkable Observations

At observing players while performing the case study the author recognized that initially, two students held a fundamental misconception regarding the *Essence Navigator* inside the *Simulation Game*. In their misconception, they would use the *Essence Navigator* to define the (virtual) “reality” of the endeavor inside the game. They thought just ticking off some Checkpoints and setting some Alpha States as reached would progress their SE endeavor inside the game. This misconception was corrected quickly, by asking them to treat the *Essence Navigator* like a map and compass not like a “transporter” in the Star Trek universe².

5.2.2.2 Game Results And Measurements

Table 5.1 on page 264 and figure 5.12 on the facing page summarize teams’ results in the case study. The average of all teams’ scores was

² According to Wikipedia “A *transporter* is a fictional teleportation machine used in the Star Trek universe. Transporters convert a person or object into an energy pattern (a process called dematerialization), then ‘beam’ it to a target, where it is reconverted into matter (rematerialization).”[304]

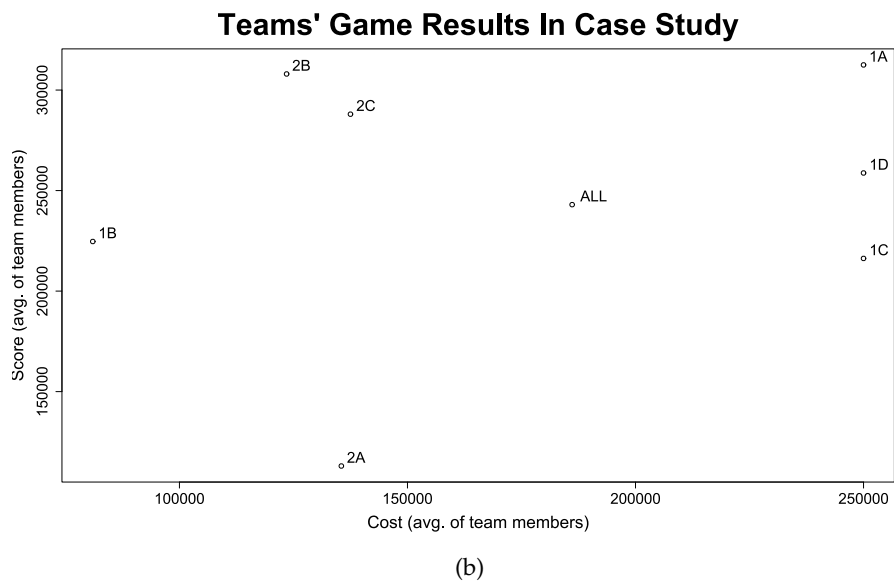
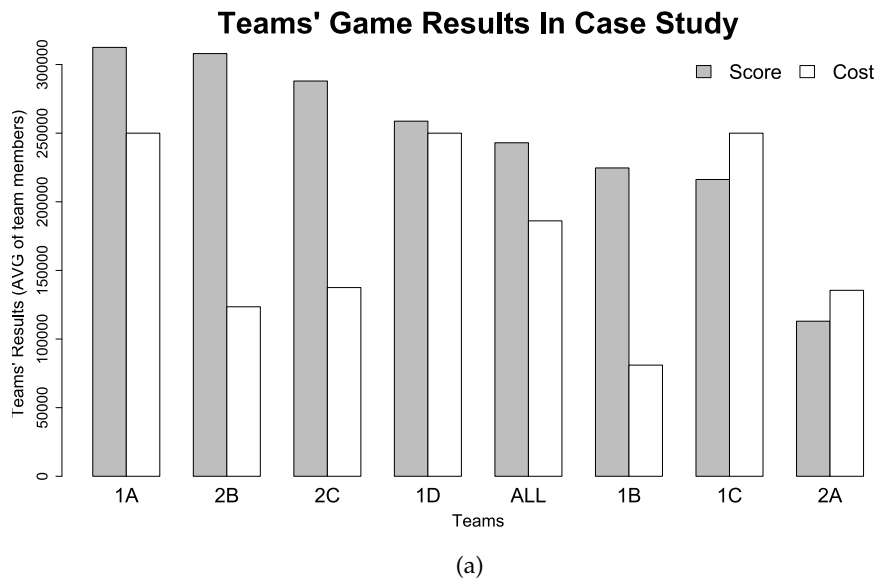


Figure 5.12: Game Results of Teams In the Case Study

Group/Team		CFP	CCP	SCORE	COST	RANKING
1A	AVG	165,500	147,000	312,500	250,000	4
	SD	6,500	6,000	12,500	0	
1B	AVG	114,667	110,000	224,667	81,000	2
	SD	6,944	6,377	13,300	2,160	
1C	AVG	136,250	80,000	216,250	250,000	6
	SD	23,742	40,330	39,111	0	
1D	AVG	140,000	118,750	258,750	250,000	5
	SD	30,895	37,466	67,229	0	
2A	AVG	91,500	21,500	113,000	135,500	7
	SD	17,500	15,500	2,000	16,500	
2B	AVG	160,000	148,000	308,000	123,500	1
	SD	19,000	17,000	36,000	50,500	
2C	AVG	148,000	140,000	288,000	137,500	3
	SD	59,000	63,000	122,000	30,500	
ALL	AVG	135,737	107,263	243,000	186,105	
	SD	34,919	50,349	78,884	72,247	
	MED	141,000	114,000	249,000	250,000	
	MAX	207,000	203,000	410,000	250,000	
	MIN	74,000	6,000	111,000	73,000	

Legend:
 CFP="Cumulated Fulfilled Checkpoints Points",
 CCP="Cumulated Consciousness Points"

Table 5.1: Game Results of Teams In the Case Study

243,000 points, with an average of 135,737 points for achieving checkpoints (CFP) and an average of 107,263 points for consciously assessing checkpoints (CCP) (cf. section 4.7.3 on page 231). These results indicate that the players were able to master the gameplay and tools provided for support. To get these points players had to progress their virtual endeavors (cf. 4.7.3 on page 231) and assess its Alphas consciously. To do that, players had to assess the current state of their endeavors based on the feedback provided by the virtual team, to plan the next target states, and to identify the next appropriate Activities (Activity Spaces) in order to assign them to their virtual teams. Players' scores prove that they were able to follow the *Plan-Do-Check-Adapt*-cycle (cf. section 2.7.2.1 on page 105), which is characteristic for driving an SE endeavor utilizing Essence and its kernel (cf. section 2.7.2 on page 97). Players had to utilize relationships between Essence kernel's elements and to show that they are already familiar with them, at least to some extent.

The results of the teams vary widely. Teams of *Group 2* spent much less money due to a shorter period of playing the game. Nonetheless, two of those teams achieved a high score (2B and 2C are both ranked in top 3). It is likely that they would have achieved even higher scores if they played for a longer period.

Teams 1A and 1B show small values for the standard deviation of both CFP and CCP. These values indicate homogeneous player strategies, which may have been facilitated by an effective team communication—or could be caused by very similar player types composing the team.

The results of team 1C show a low value for the average but a high value for the standard deviation of the CCP. This indicates that not all players of the team were consciously assessing their Alpha States based on the feedback provided by the virtual team. More, or better, team communication and alignment of individual results with the ones of the teammates probably would have provided even better scores.

Team 2C provides the highest values of the standard deviation for CFP, CPP, score, and cost. These values indicate team dynamics like in real software projects. The team members of team 2C were those that did know each other the least. Observations at conducting the case study showed only occasional communication between team members. As we know, e.g. from studies of Tuckman and Jensen[287], teams need time to get to a high *performing* state³.

These results strongly indicate that providing the opportunity to play as a team may work well, as esp. shown by teams 1A and 1B, but does not guarantee such desired cooperation. Depending on the initial situation, the short period allocated to playing the game in the case study may not have provided enough opportunity to get to that

³ This is represented by the states of the *Team Alpha* in the Essence Kernel as well.

Group/Team	Ratio of players using time travel	time travels			days (time) traveled		
		Count	AVG*	SD*	Count*	AVG*	SD*
1A	0%	0	0	0	0	0	0
1B	67%	3	1.5	0.5	42.0	21.0	6.3
1C	50%	30	15.0	14.0	181.0	90.5	82.4
1D	50%	5	2.5	0.5	118.0	59.0	33.3
2A	67%	7	3.5	0.5	69.0	34.6	0.7
2B	67%	12	6.0	1.0	294.4	147.2	96.2
2C	100%	38	19.0	1.0	105.3	52.7	25.3
ALL	63%	95	7.9	8.8	809.8	67.5	68.6

* based only on players using the *Time Travel* feature

Table 5.2: Utilization of the Time Travel Feature by Teams

state. Team dynamics while playing the Simulation Game provide a very interesting field of research for future work.

Table 5.2 summarizes the utilization of the *Time Travel* feature by the teams. This feature was designed to provide the opportunity to *learn from failures* (cf. section 4.7.3 on page 231). Overall the players of the case study traveled 95 times in time and covered 810 days thereby.

We can see that the usage of this feature varies widely and ranges from 0% to 100% of team members utilizing this feature. Teams 1A and 2C provide interesting extreme manifestations of this issue. While none of the members of team 1A did use the time travel feature, all players in team 2C used it with the highest count of performed time travels. Both teams achieved high scores—at different efficiency.

While members of team 1A tried to avoid making mistakes by carefully observing the feedback provided by the virtual team, members of team 2C additionally utilized the *Time Travel* feature to improve achieved scores iteratively. The low value of the difference between the average CFP and CPP indicates a very conscious assessment of Checkpoints and Alpha States. *Time Traveling* providing to repeatedly experience the same situation in the game very likely supported at achieving this result.

Players of teams 1B and 2B seem to have combined both strategies with success too.

By providing the smallest difference between the average CFP and CPP values, combined with only a small count of performed time travels, players of team 1B achieved scores indicating a very conscious playing strategy with the homogeneous performance of team

members. This is indicated by small values of standard deviation of CFP and CPP, at a slower progression of the game, which is indicated by relatively small cost. Members of this team were observed to be communicating very actively.

Players of team 2B utilized the *Time Travel* feature the most in terms of days that were traveled in time. This strategy contributed to achieving a high score combined with lower cost and resulted in the best-ranking score.

These numbers may indicate different kind of players and learners, e.g. players that aim at performing without mistakes right from the start, much like in a real SE endeavor, and others, which are willing to add the given opportunity to iteratively improve results.

One player of team 1C showed the massive utilization of the *Time Travel* feature and traveled 29 times covering 173 days. Combined with a low score for CPP this indicates a rather unconscious try and error strategy arguing for the importance of feedback provided by the team and debriefing activities in order to facilitate reflection and learning.

The mixed rankings of *Group 1* and *Group 2* indicate that the *Simulation Game* provided the opportunity to learn for both the more and the less experienced student group. *Group 2*, the more experienced students, were not able to outperform *Group 1* in the setting of this case study.

5.2.2.3 Questionnaire Results

Students playing the Simulation Game have fun (Sign-test, $\alpha = .05$, cf. D.1 on page 370) and attribute learning success among others to it (cf. table 5.4 on page 279). Figure 5.13 on the next page shows the results regarding the fun perceived by students playing the simulation game. 57.9% ($n=11$) of the participants reported that they had fun playing the game, while 42.1% ($n=8$) chose the neutral answer item.

The difficulty of the Simulation Game was perceived as “just right” by 42.1% ($n=8$) of the participants. 26.3% ($n=5$) found it slightly easy and 31.6% ($n=6$) of the players found it slightly hard. None of the students assessed the game as either “very easy” nor “very hard.” This data is presented in figure 5.14 on page 269.

Figure 5.15 presents how players of the Simulation Game experienced the duration of the game. The game was designed to enable players to achieve (at least most of) intended results in a standard 90 minutes course unit. Players were given the opportunity to continue to play the game outside of the regular course units till the next regular course meeting (one week later) where the results of the game were summarized and discussed in a debriefing activity. The vast majority (78.9%, $n=15$) of all students assessed the duration as “just right,” 3 players (15.8%) felt the duration was “too long” and only one player (5.3%) felt it was “too short.”

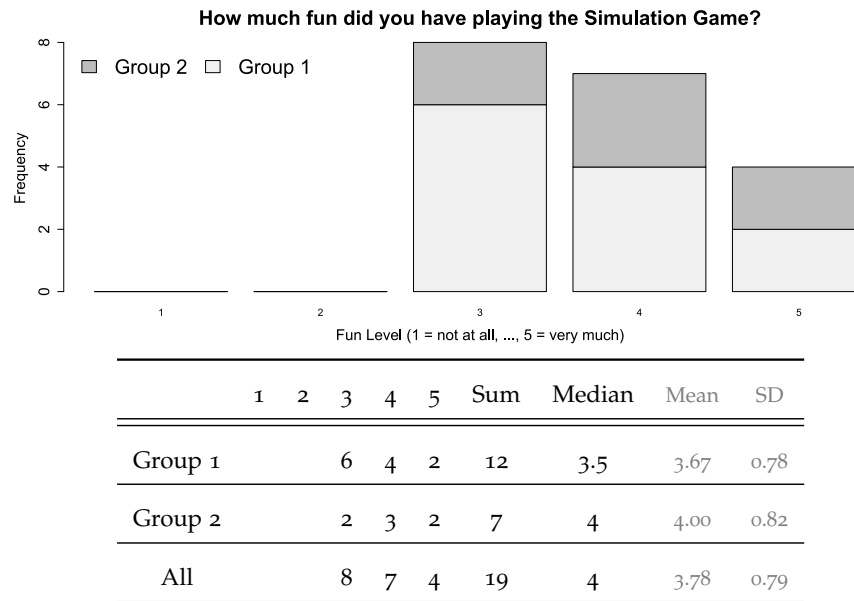


Figure 5.13: Q09: How much fun did you have playing the Simulation Game?

89.5% ($n=17$) of all participants stated they would play the Simulation Game again (cf. figure 5.16 on page 271) and students would recommend the game to a friend or fellow student (Sign-test, $\alpha = 0.05$, cf. D.2 on page 371 and figure 5.17 on page 271).

Students stated they would play the game again to optimize their performance and to improve their personal high score. They would like to benefit from experiences of the first run and to test findings in a next one. They stated, they would like to internalize proceedings further, because learning was perceived faster and deeper as in lectures, and finally, because it was perceived as fun.

Students recommend the simulation game as standard part of SE courses (Sign-test, $\alpha = 0.05$, cf. table D.1 on page 372 and figure 5.18 on page 272) Asked if this game should be a standard part of any SE course, 95% ($n=18$) of the participating students answered positively. Only one student of *Group 1* chose the neutral item.

Asked for the reasons, why they recommend the Simulation Game as a standard part, participants stated, that they appreciated the innovative and creative learning approach actively supporting the learning focused on essential content, the relatively short duration of the game, the overview and preparation of upcoming projects provided. The students liked that relationships between elements, e.g. Activities and Alphas, got visible and tangible. Students mentioned fast feedback, focus on virtually practicing steps in an endeavor and the opportunity to learn from failures as particularly advantageous. Here are three statements of students:

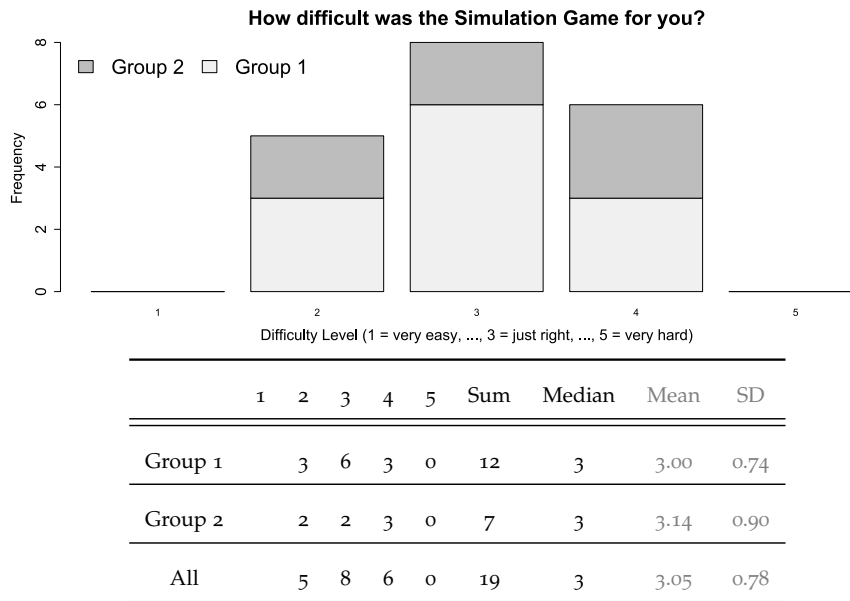


Figure 5.14: Q10: How difficult was the Simulation Game for you?

- “The project seems too comprehensive for absolute novices to utilize Essence in a reasonable way. The game helps to better understand successive working steps. For the remaining semester, the game provides a good starting point to the real project work.”
- “The game is a good way to learn relationships of single areas fast and structured. Everyone can make own experiences at his own pace. Pure lectures would hardly provide that and rather deliver just new terms hard to digest. To provide steps to do in proper sequence in a presentation seems hard too. The game provides an opportunity of trying out the concepts. ”
- “Moreover one can—unlike as in a real project—make mistakes and learn from them.”

Students (95%, $n=18$) stated that they learned something about SE by playing the game. Interestingly the more experienced students of *Group 2* perceived occurred learning even stronger than those of *Group 1* (cf. figure 5.19 on page 272). 86% of *Group 2* chose the answer item representing highest agreement.

79% of the participants ($n=15$) stated that they reinforced SE knowledge of prior lectures/courses. Two participants stated that they did not reinforce prior knowledge at all. One of them stated not to have visited the course very regularly.

Students stated, they perceived that they learned something new about SE by playing the Simulation Game (Sign-test, $\alpha = 0.05$, cf. figure 5.21 on page 273 and table D.1 on page 372). Asked, what students learned new by playing the game, they stated:

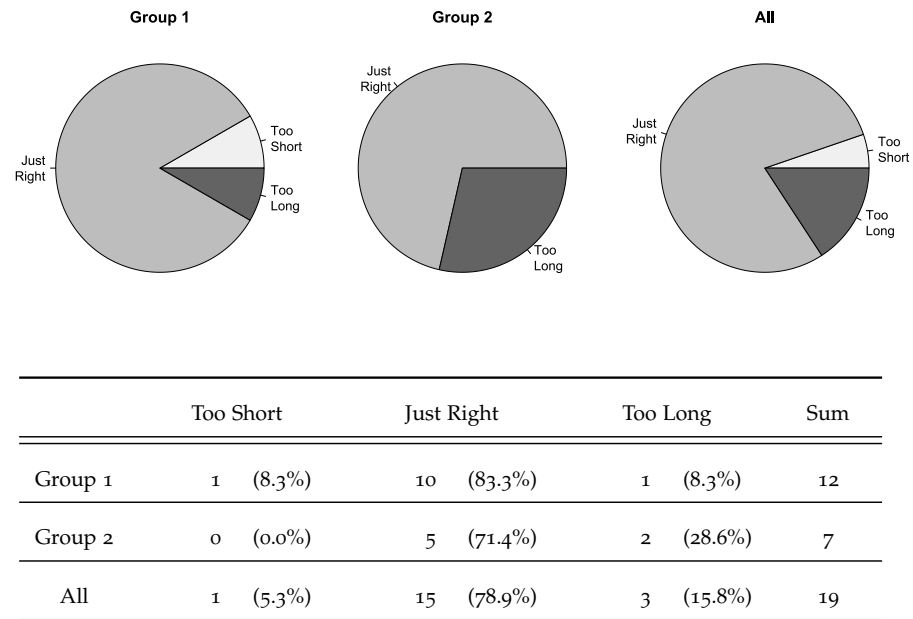


Figure 5.15: Q11: How was the duration of the Simulation Game in your opinion?

- an impression of the multiple aspects to control and think about at running a software project,
- the complete Essence Kernel,
- how to deal with Alphas,
- the logical sequence of steps at Alphas' assessment,
- Essence is not an abstract construct anymore,
- an impression of how to proceed, and
- the relationships between a software project and its elements (Alphas, Alpha States, Checkpoints, and Activities).

The participants stated that having to think about the next steps to take actively and to having to repeatedly process Alphas and their Checkpoints, to recurrently think about them in detail, helped to recognize relationships between elements and to internalize concepts.

Students perceived to have improved their performance while playing the Simulation Game (cf. figure 5.22 on page 276 and table D.1 on page 372). This seems to be a good indicator of felt self-efficacy and fits well to the perceived learning.

Table 5.3 on page 278 and table 5.4 on page 279 summarize students answers to questions, what helped them most to improve their performance inside of the game and to learn by playing the Simulation Game.

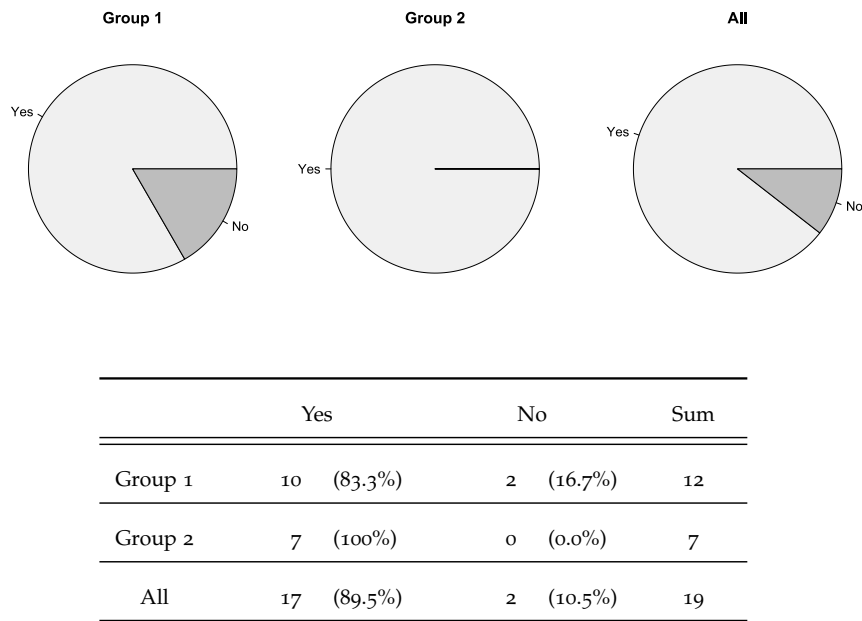


Figure 5.16: Q17: Would you play the Simulation Game again?

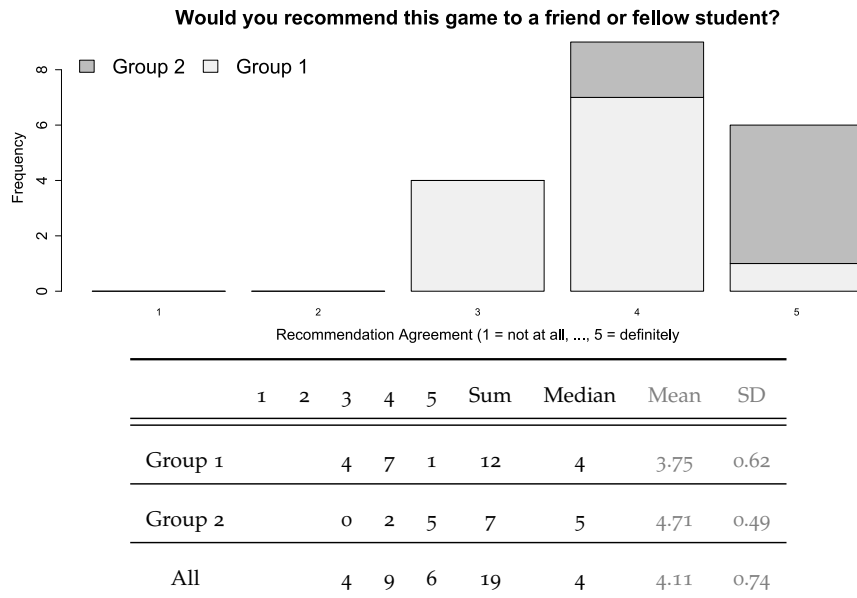


Figure 5.17: Q19: Would you recommend this game to a friend or fellow student?

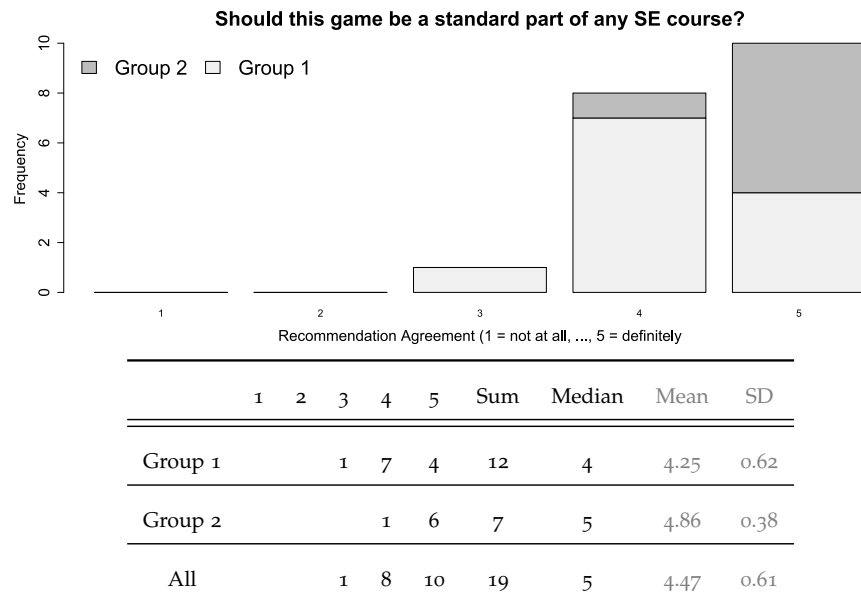


Figure 5.18: Q20: Should this game be a standard part of any SE course?

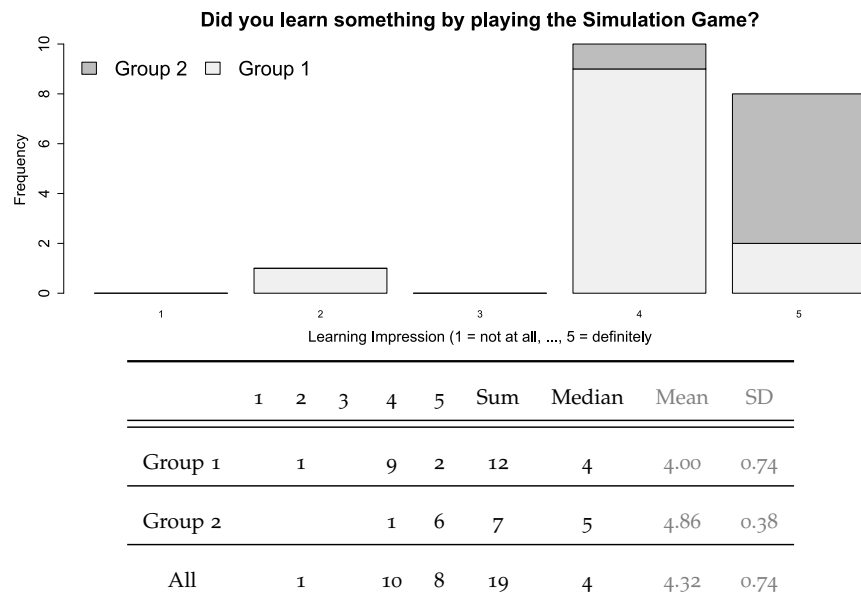


Figure 5.19: Q22: Did you learn something about SE by playing the Simulation Game?

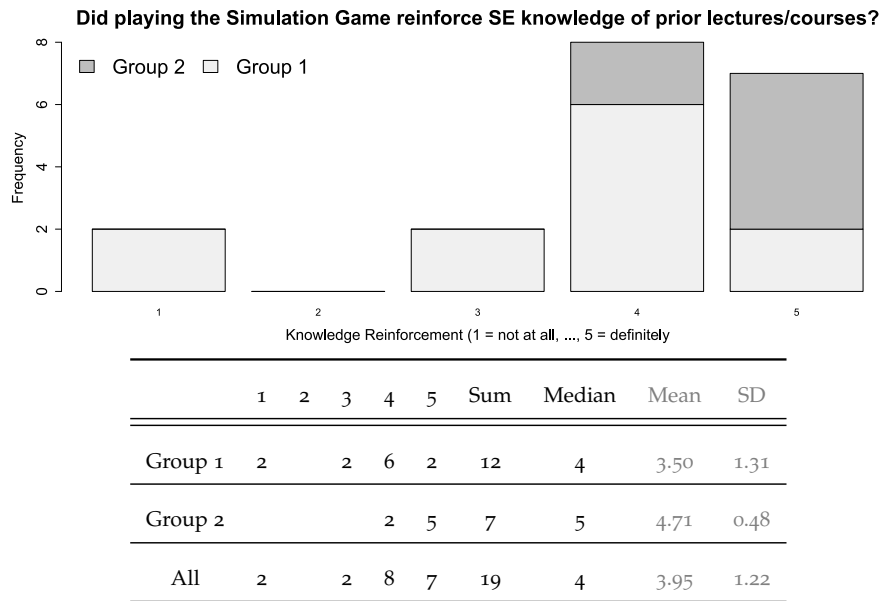


Figure 5.20: Q23: Did playing the Simulation Game reinforce your SE knowledge of prior lectures/courses?

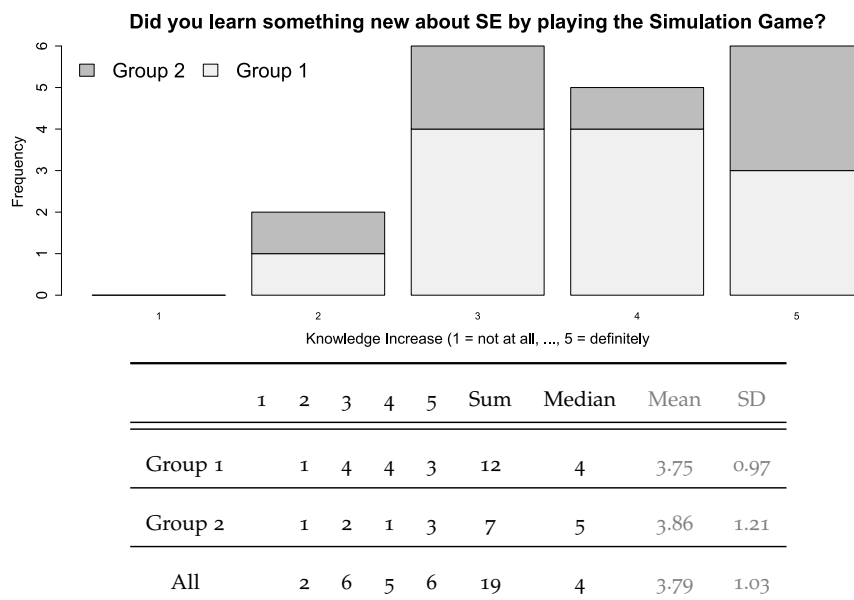


Figure 5.21: Q25: Did you learn something new about SE by playing the Simulation Game?

All (100%, $n=19$, answer options 2 and 3) students stated that the fun at playing the game, the necessity to make own active decisions, and the Essence Navigator helped at learning. None of the students stated that one of these features did not help at learning. Of these features *active decisioning* was assessed as very strongly helping by the highest number of participants (79%, $n=15$ | Group 1: 75%, $n=9$ | Group 2: 86%, $n=6$) followed by fun (74%, $n=14$ | Group 1: 75%, $n=9$ | Group 2: 71%, $n=5$).

These results indicate strong arguments for *DGBL* and inherent *constructivist learning* approaches (cf. section 2.2.3 on page 24). That the *Essence Navigator*, providing *scaffolding* (cf. section 2.2.3 on page 24) by embedding learning content in the form of Essence kernel's elements with short descriptions, providing orientation (if used reasonably), and visualizing the progress made, was perceived as that helpful, confirms the assumption that digital learning games should embed learning content about the process, method, or kernel in this case. This was an aspect identified as lacking by most of the existing *DGBL* approaches so far (cf. section 2.5 on page 68).

Students stated that *debriefing activities* (89%, $n=17$), as well as *team communication* (84%, $n=16$), provide learning. Out of them almost 1/3 of the students indicated these two aspects as *very strongly supporting* learning, while the remaining 2/3 considered them as helpful. The elementary team chat provided by the game environment was not heavily utilized. As one student stated: "We had no need for the team chat because we were sitting next to each other." The author of this thesis noticed active direct conversations between team members while playing the game. So it can be stated that the game provided triggers for social interaction, but players preferred face-to-face communication instead of chatting. This was a very welcome effect since face-to-face talks are a much richer way of communication than pure text chat.

These results indicate strong arguments for *social constructivist* learning approaches (cf. section 2.2.3 on page 24).

While the features above all showed statistically significant support to learning (Sign-test, $\alpha = 0.05$, cf. table 5.4 on page 279 and table D.1 on page 372), the *time travel* feature could not prove that at a significance level of $\alpha = 0.05$ (cf. table D.1 on page 372). This feature was built in to provide an opportunity to *learn from failure*. It seems that not all students recognized this feature as supportive as expected. 63% ($n=12$) of participating students stated that this feature was helpful at learning. One-half of them perceived this feature as very strongly helping. This assertion is supported by the free text answers stating several times that students appreciated the "opportunity to learn from mistakes."

Table 5.2 summarizes the utilization of the time travel feature by players and teams. These results show that only 63% of all players used this feature. This value of 63% matches the ratio of players appreciating the time travel feature as supportive of learning. Since the answers to the questionnaire were collected anonymously, no clear correlation can be determined, but the results may indicate that players using this feature appreciate it as supportive of learning.

Other features built into the simulation game that were *not* able to show statistically significant support to learning are *rankings and leaderboards* (Sign-test, $\alpha = 0.05$, cf. table 5.4 on page 279 and table D.1 on page 372). Only 42% ($n=8$) of participating students stated that rankings and leaderboards provided support at learning, 58% of the students ($n=11$) did not.

These features were built into the Simulation Game to provide occasions and triggers for social interaction inside the teams. Showing how well the own performance is in relation to other ones inside the team and how well the team performs compared to other teams should provide triggers to reconsider, reflect and discuss.

As students agreement to helping team communication indicates, the game provided triggers for interaction. Why rankings and leaderboards were not perceived supportive by the majority of players might have several reasons and be caused by the dynamics of the gameplay of the teams.

As observations at conducting the case study showed, players explore and perform the game at different speed. In such a setting fast team members ahead provide aligning rankings to the players following but do not yet have other one's results to compare the own performance to. In such cases, the rankings and leaderboards do not unlock their potential to "fast-forward" players and may be not perceived as really supportive.

Once team communication got started and had been in a rather ongoing state, not all team members had to look at the leaderboards provided. Results may be compared as part of verbal team communication too. Such assumption is supported by the observation that ten out of the eleven players not valuing rankings and leaderboards assessed team communication as supportive to learning.

Overall these results provide interesting fields of observation and research for future studies of the game. It seems to be an interesting task to prove or disprove the hypothesis that players, assessing time travels as well as rankings and leaderboards as not supportive to learning, draw primarily from team's experience and team communication—and maybe trying to avoid failures already made by others. To support such findings, the results of the game and the questionnaires would need some mapping—something that was not implemented in this first study.

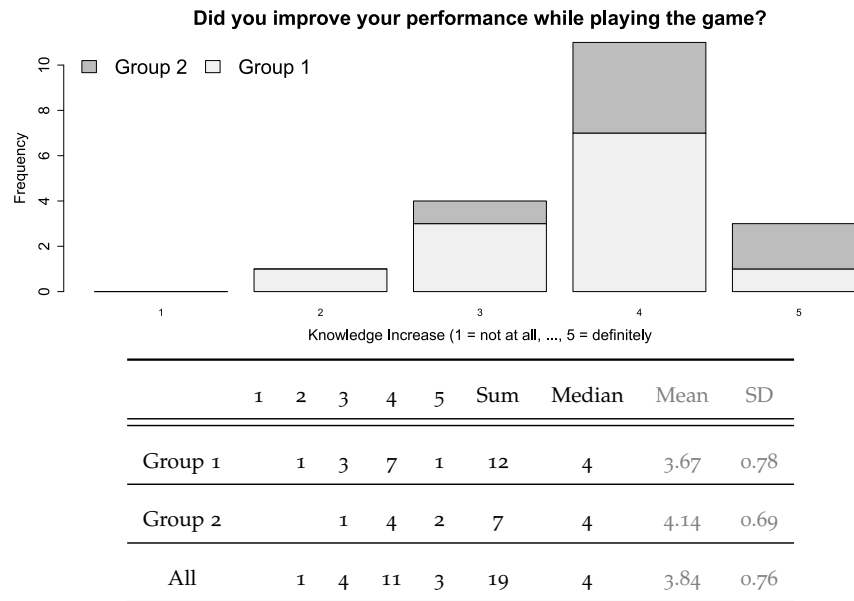


Figure 5.22: Q27: Did you improve your performance (higher score, less costs) while playing the Simulation Game?

Students stated that they did rather not feel familiar with SEMAT Essence concepts before playing the Simulation Game (cf. figure 5.23 on the facing page). Most of them (58%, $n=11$) chose the neutral answer item, and another 32% ($n=6$) chose the *less familiar* item.

After playing the Simulation Game, students felt more familiar with the SEMAT Essence kernel. Only one student chose the neutral answer item. None of the participants chose a lower rated answer item. 95% of the students stated to feel *rather familiar* with Essence concepts at this point. None of the students chose the highest rated answer item representing to feel already *very familiar* with Essence.

These answers let assume that while students gained familiarity, they were not misled “to know everything there is” but recognized that there is even more to learn.

This shift in perceived familiarity also supports the assumption that students learned something new by playing the game.

5.2.3 Essence Navigator

This section presents results of the questionnaire with regards to the Essence Navigator. Out of the 19 students, 18 (95%) stated that they would like to deploy a tool like the *Essence Navigator* in one of their future projects (cf. figure 5.25 on page 280). Asked for their reasons they stated that

- the tool provides a “good help at structuring,”

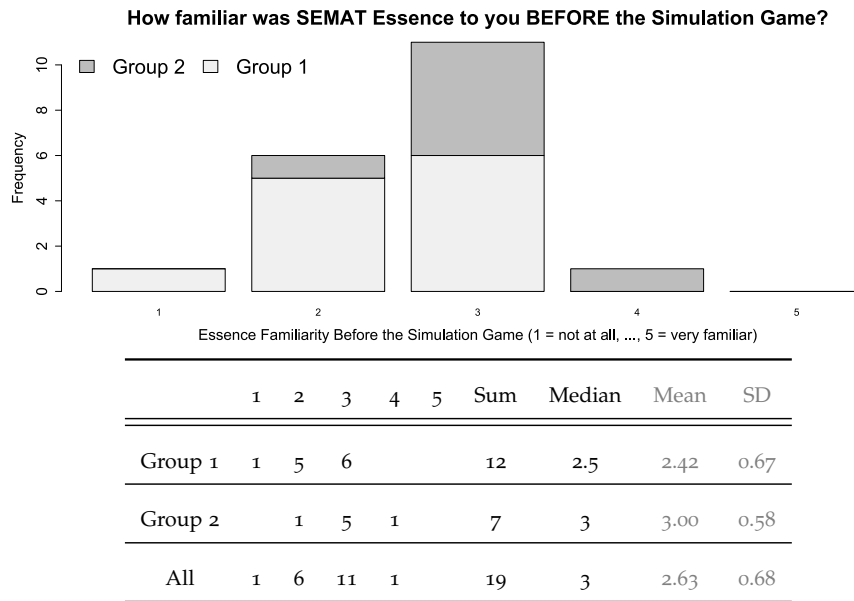


Figure 5.23: Q42: How familiar was SEMAT Essence to you BEFORE the Simulation Game?

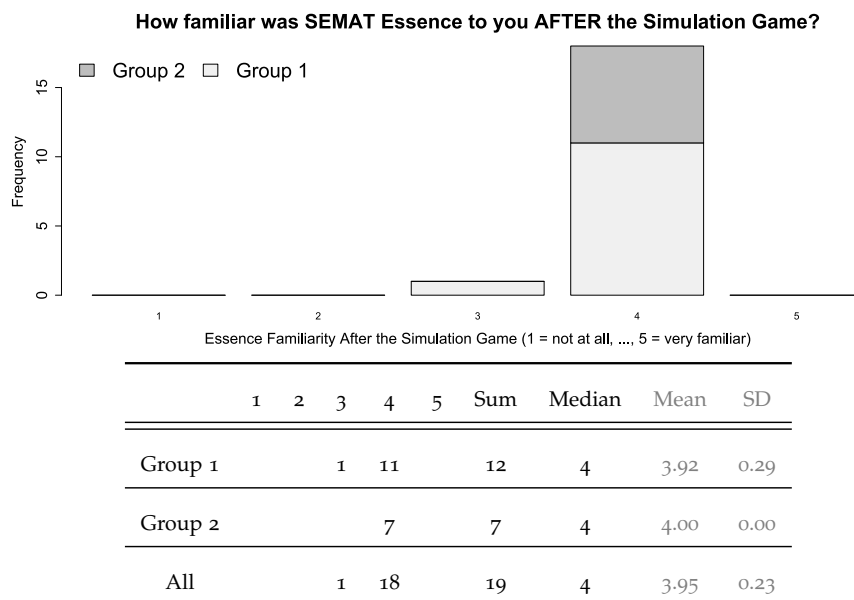


Figure 5.24: Q43: How familiar was SEMAT Essence to you AFTER the Simulation Game?

#	Group	Improvement Supported by											
		TC			TT			R/L			NAV		
		1	2	3	1	2	3	1	2	3	1	2	3
1	1		X			X				X		X	
2	1			X	X			X				X	
3	1	X			X			X					X
4	1			X		X				X			X
5	1			X		X		X					X
6	1			X		X		X				X	
7	1			X		X		X				X	
8	1			X			X	X					X
9	1		X			X		X					X
10	1	X					X		X			X	
11	1	X			X				X				X
12	1		X		X			X					X
13	2			X			X		X				X
14	2		X				X			X		X	
15	2		X		X					X		X	
16	2	X				X			X			X	
17	2		X			X				X			X
18	2		X				X	X					X
19	2			X			X	X					X
Σ_{G1}		3	3	6	4	6	2	8	2	2	0	5	7
%G1		25	25	50	33	50	17	67	17	17	0	42	58
		25	75		33	67		67	34		0	100	
Σ_{G2}		1	4	2	1	2	4	2	2	3	0	3	4
%G2		14	57	29	14	29	57	29	29	43	0	43	57
		14	86		14	86		29	72		0	100	
Σ_{G1UG2}		4	7	8	5	8	6	10	4	5	0	8	11
		4	15		5	14		10	9		0	19	
%G1UG2		21	37	42	26	42	32	53	21	26	0	42	58
		21	79		26	74		53	47		0	100	
Legend: TC="Team Communication", TT="Time Traveling", R/L="Ranking/Leaderboards", NAV="Navigator", ordinal scale {1,2,3}, 1="not at all", 2="less strong", 3="very strong"													

Table 5.3: Q28: What helped you the most to improve your performance while playing the Simulation Game?

#	Group	Learning in Simulation Game Supported by																				
		TC			TT			R/LB			NAV			DEB			AD			FUN		
		1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
1	1		X				X			X		X			X				X			X
2	1			X	X			X				X			X			X			X	
3	1	X			X			X					X		X			X			X	
4	1			X		X			X			X				X			X			X
5	1			X		X		X					X		X			X			X	
6	1			X		X		X				X				X			X			X
7	1		X		X			X				X			X				X			X
8	1		X				X	X					X	X					X			X
9	1		X		X			X				X				X			X			X
10	1	X				X			X			X			X				X			X
11	1	X			X					X			X		X				X			X
12	1		X		X			X					X			X			X			X
13	2			X			X		X				X			X			X			X
14	2		X				X		X				X		X				X			X
15	2		X		X			X					X		X				X			X
16	2		X			X			X			X			X				X			X
17	2		X			X				X			X	X				X			X	
18	2		X				X	X					X		X				X		X	
19	2			X			X	X					X			X			X			X
Σ_{G1}		3	5	4	6	4	2	8	2	2	0	7	5	1	7	4	0	3	9	0	3	9
%G1		25	42	33	50	33	17	67	17	17	0	58	42	8	58	33	0	25	75	0	25	75
		25	75		50	50		67	34		0	100		8	91		0	100		0	100	
Σ_{G2}		0	5	2	1	2	4	3	3	1	0	1	6	1	4	2	0	1	6	0	2	5
%G2		0	71	29	14	29	57	43	43	14	0	14	86	14	57	29	0	14	86	0	29	71
		0	100		14	86		43	57		0	100		14	86		0	100		0	100	
Σ_{G1UG2}		3	10	6	7	6	6	11	5	3	0	8	11	2	11	6	0	4	15	0	5	14
		3	16		7	12		11	8		0	19		2	17		0	19		0	19	
%G1UG2		16	53	32	37	32	32	58	26	16	0	42	58	11	58	32	0	21	79	0	26	74
		16	84		37	63		58	42		0	100		11	89		0	100		0	100	
Legend: TC="Team Communication", TT="Time Traveling", R/L="Ranking/Leaderboards", NAV="Navigator", DEB="Debriefing", AD="Active Decisioning", FUN="Fun at Playing"																						
ordinal scale {1,2,3}, 1="not at all", 2="less strong", 3="very strong"																						

Table 5.4: Q29: What helped you the most to learn while playing the Simulation Game?

Feature	S_-	S_+	pValue	Accepted	Rejected
Team Communication	3	16	.002	H_1	H_0
Time Travel	7	12	.180	H_0	H_1
Ranking/Leaderboards	11	8	.820	H_0	H_1
Navigator	0	19	.000	H_1	H_0
Debriefing	2	17	.000	H_1	H_0
Active Decisioning	0	19	.000	H_1	H_0
Fun	0	19	.000	H_1	H_0

(Sign - test, $H_0 : \eta \leq 1.5$, $H_1 : \eta > 1.5$, $n = 19$, $\alpha = .05$)

Table 5.5: Summary of Hypotheses Testing Regarding Features of the Simulation Game obtained by Sign-tests

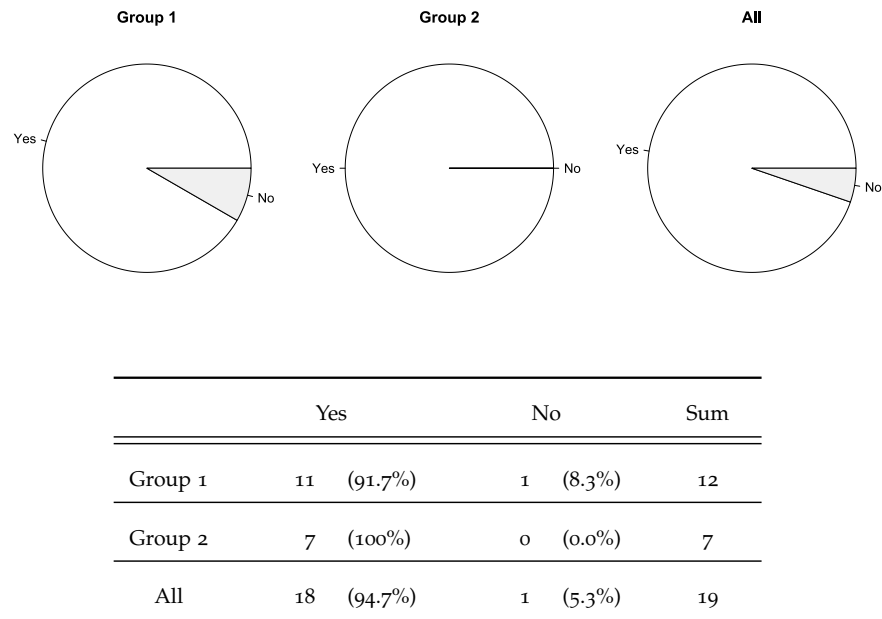


Figure 5.25: Q39: Would you like to deploy the Essence Navigator in one of your future projects?

- it helps to better visualize the progress and current state of the project,
- it provides a quick overview of the current state,
- it supports at smoothly defining next goals and steps to take, and
- it helps not to forget something.

Obviously, most of these mentioned advantages refer to Essence itself, but the answers indicate that the provided *Essence Navigator* did not hide those beneficial characteristics.

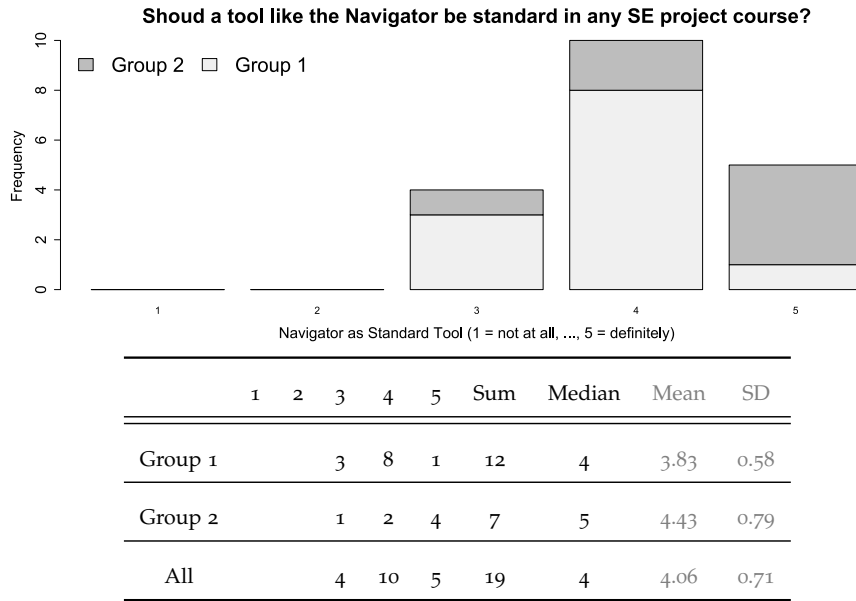


Figure 5.26: Q40: Should a tool like the Navigator be standard in any SE project course?

In responding students opinion, a tool like the *Essence Navigator* should be set as standard in any SE project course (Sign-test, $\alpha = 0.05$, cf. figure 5.26 and table D.1 on page 372).

5.2.4 SEMAT Essence

Responding students stated that they think, SEMAT Essence and the Essence Kernel will be of help in their future projects (Sign-test, $\alpha = 0.05$, cf. figure 5.27 on the following page and table D.1 on page 372). None of the students stated that they think it would not provide support.

Students stated that they want to deploy SEMAT Essence in their future projects (Sign-test, $\alpha = 0.05$, cf. figure 5.28 on the following page and table D.1 on page 372). None of the students stated not to want to deploy Essence in future projects. A third ($n=6$) of all participating students chose the neutral item at answering this question.

5.2.5 Integrated Approach

While the preceding sections summarized the results of single tools and aspects, this section provides a summary of the *Integrated Approach* as a whole. The phases of this approach got described in chapter 4 on page 179. As part of the conducted case study, the first three phases got performed. These three phases covered aspects of most undergraduate curricula including course or capstone projects.

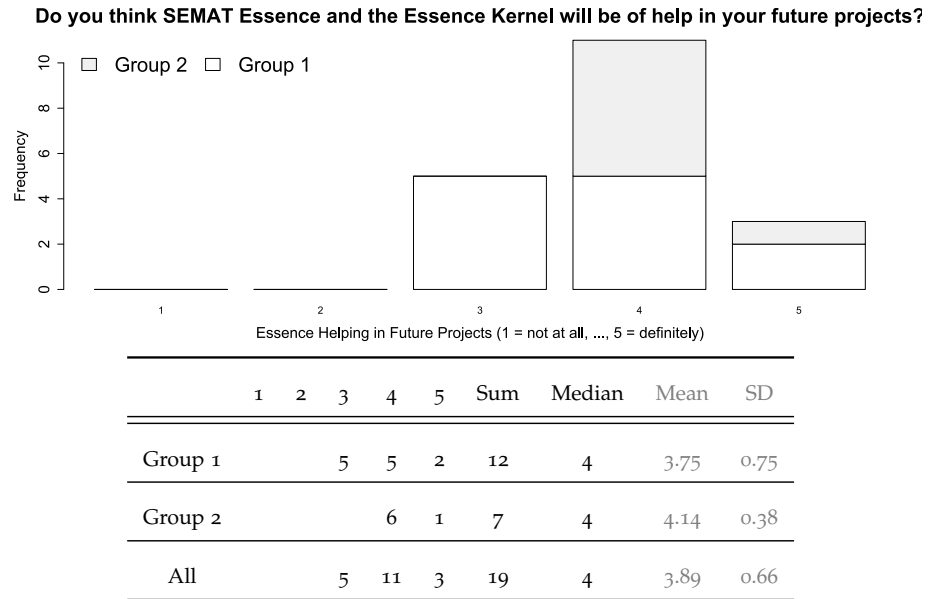


Figure 5.27: Q44: Do you think SEMAT Essence and the Essence Kernel will be of help in your future projects?

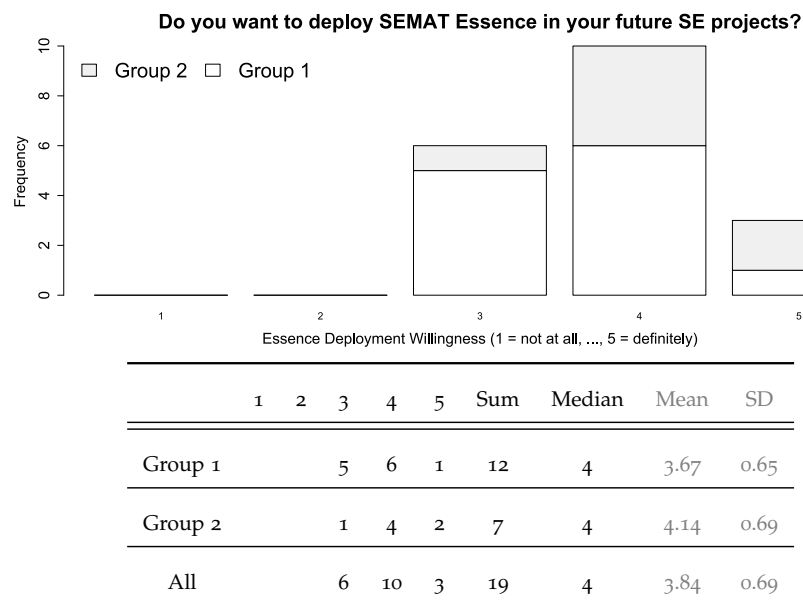


Figure 5.28: Q45: Do you want to deploy SEMAT Essence in your future SE projects?

The procedures conducted were already described in detail in section 5.1 on page 251. Only the treatment of *Group 1* of the case study followed the first three phases of the proposed *Integrated Approach*. *Group 2* only used single parts of it and provided feedback to the tools provided and tested by them.

The presented results of the first phase (cf. section 4.5.5 on page 209 and section 5.2.1 on page 258) and second phase (cf. section 5.2.2 on page 262) indicate that students appreciated the approaches taken, and that the use of the *Essence Kernel Puzzler* was considered to prepare participants well for the following *Simulation Game*.

In the weeks following the *Simulation Game*, students of *Group 1* utilized the *Essence Navigator*, provided in addition to a given SE process, to drive their project work (phase 3 of the *Integrated Approach*). Alpha States and their Checkpoints were assessed in an ongoing fashion as part of regular mandatory team meetings, hold two times a week.

The *Simulation Game* provided feedback of the virtual team that was quite easy to map to Checkpoints of an Alpha State. While working on their own (real) projects the teams had to provide that kind of feedback on their own. The Alphas, their Alpha States, and Checkpoints provided valuable triggers for thinking about the project and guidance in team discussions.

Corresponding to existing findings of other deployments of the Essence kernel[207], some ambiguous formulations and single terms used in the descriptions of checkpoints provided reasons for requesting and discussions. Péraire and Sedano[207] reported that these ambiguities were leading to situations “where the team discusses the meaning of a checklist item instead of having a conversation about the project.” In the context of the course project, aiming at introducing students into SE practices and methods, such discussions were not judged as impeding but welcome since they provided occasions to reflect on approaches prescribed by the process provided and to think about alternatives.

Since students of *Group 1* mainly had no chance to develop more complex software systems, they were not able to draw on any existing experiences utilizing SE practices covering the essential aspects of an SE endeavor (cf. figure 2.14). That is why they were provided with an SE process tailored to the need of the course. Since the given SE process was not implemented using the Essence language, some concepts had to be mapped and discussed. For instance, the given SE process made use of *Use Cases* to structure requirements. The Essence kernel on its own does not provide substructures of requirements since this is handled by different practices in various ways. This challenging

aspect was reported by Péraire and Sedano[207] too. Differing from their study students in this case study did not already have a considerable average work experience in industrial practice.

Despite that, students were able to utilize the provided enacted Essence kernel and the already familiar *Essence Navigator* largely right from the start of the project work and without further support by the lecturer—hence they showed not just that they *understood* concepts but proved the ability to already *apply* the learned concepts. This indicates that the *Simulation Game* contributed to lowering the *cognitive load* that students faced in their project work. Following the *cognitive load theory (CLT)*, any knowledge already learned, in terms of CLT stored in long-term memory, frees room in the rather limited working memory and facilitates learning of related new knowledge (cf. section 2.2 on page 19). This indicates too that the *Simulation Game* provided a valuable *anchor* that students were able to refer to in their thinking and discussions with teammates (cf. *Anchored Instruction* in section 2.2 on page 19).

At some points and depending on individual working style, some teams had slightly more a tendency to quickly answer questions and assess Checkpoints in a rather superficial way than others. Based on limited experience developing more complex software systems, this comes as no real surprise and may reflect the approach of any novice acting in SE projects. By asking to explain the reasoning behind the assessment of Checkpoints and Alpha States, those teams were motivated to think further about specific issues.

In an optional exercise, students were asked to map explicitly

- the assigned activities of the given SE process to Activity Spaces of the Essence Kernel and
- the points in time, when single Alpha States got achieved, to phases and concrete iterations of the given software process.

Alpha States that were not part of the assigned project work had to be marked. At each mapping, the students were asked to provide a short explanation of the specific mapping done.

Three out of four teams performed this exercise and delivered encouraging reasonable results. As students perceived the activities to do, beside designing and implementing source code, as very comprehensive, they were surprised that some essential aspects of an SE endeavor were not covered by their assigned project work for organizational reasons of the course. Those aspects got clearly visible by non-addressed Alpha States and put the project work done into perspective.

Of course introducing additional concepts like Essence consumes time. Alpha assessment sessions as part of regular team meetings consume time too. Without time added to a course, the time available for remaining tasks is inevitably reduced. Students in the case study were able to deliver their projects with at least the same quality as teams on the same course in years before. To be able to get familiar with the additional concepts, some of the features of the required software project outcome were omitted. Those features would have provided additional functionality to the resulting software but conceptionally had repeated lessons already learned by the students in slightly different ways.

Altogether it can be summarized that the working style of the teams in the case study was much more oriented towards a holistic perspective of the SE endeavor than the working style of teams in the years before, where the same course was held.

Teams discussed much more about essential aspects that rather got ignored in the past where only the SE process was given to the teams.

Students always were able to communicate the current state of their SE endeavor and next steps to take. The former wide-spread quick lookup at the provided course schedule and deadlines set, to infer the current state of the project, or rather the current desired condition of the project, was replaced by a structured and goal-oriented approach that was appreciated by students of the course.

Thereby the teams did not loose the focus on operative SE tasks to do in the process but were additionally able to give reasons for doing those tasks. Utilizing the generalist approach of the SEMAT Essence Kernel students were provided with a highly transferable thinking framework acting as an anchor and providing support, orientation, and guidance in any future SE endeavor.

All in all, students applied a systematic, disciplined and quantifiable approach to the development of their software—and appreciated the guidance given by that approach. Hence they acted as software engineers-to-be⁴. They showed competencies that the introduced *Integrated Approach* aimed to address (cf. section 4.1.3.1 on page 186).

The presented results indicate that the *Integrated Approach* utilized in the case study successfully addressed identified gaps to close described in section 1.3.

Furthermore, the mapping of its characteristics to

⁴ IEEE/ACM define *Software Engineering* as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software [...]”[112]

- quality attributes collected in heuristics[74] (cf. section C on page 361),
- recommendations based on the synthesis of a comprehensive SLR[126] (cf. section C.3 on page 366), as well as
- SE education requirements based on conclusions of Boehm[27] reviewing SE in the 20th and 21st century (cf. section C.2 on page 366)

indicate that the *Integrated Approach* is contributing to current SE education.

Could this approach claim to be not only effective but to be efficient too? To claim such benefits, some comparative experiments should be conducted, comparing different kinds of treatments to experiment groups. Section 5.1.4 on page 255 describes, why such an approach was not chosen and utilized for this research work. As already stated, this may follow in future research.

Although such claims cannot be made at the moment, the author of this thesis submits that only a few hours, invested in phase 1 and phase 2 of the introduced approach, enabled the opportunity to experience orientation and guidance given by provided SE concepts and tools right from the start of project work. This is increasing the likelihood that students really appreciate them and develop inner confidence of their utility.

SUMMARY

The industry, as well as curriculum guidelines, demand strong competencies in the field of SE methods. Unfortunately, this field is less intuitive accessible. Observations show that the holistic perspective on SE endeavors gets frequently lost by students in course projects. It is not easy for them to orientate inside of a given SE method and to utilize it in a systematic goal-oriented way. To establish the ability to keep a holistic view on their SE endeavors and to appreciate the orientation and guidance provided by SE methods, students have to be provided with adequate learning experiences enabling the development of SE attitudes, where the provided practices and methods are not just perceived as further cognitive load but as actually supporting at keeping a holistic perspective on the SE endeavor.

The *Integrated Approach* proposed in this thesis, introduces students to SE methods. It was designed explicitly to facilitate project-based learning approaches by lowering the cognitive load students face at conducting their project work through preparing them with Digital Game-Based Learning activities where they experience SE concepts in a hazard-free challenging environment. The introduced approach, deeply grounded in learning theories, integrates developed tools and games into a number of phases, each addressing its respective learning objectives.

To promote the transferability of competencies gained, the generalist SEMAT Essence Kernel approach, considering all essential dimensions of SE endeavors in a practice and process independent way, gets utilized. A mapping of its characteristics to demands of curriculum guidelines, as well as learning theories, proved its utility in SE education.

The evaluation of the introduced *Integrated Approach* and its components indicates that the *Integrated Approach* is contributing to the achievement of stated learning objectives and current SE education in general. Students participating in a cases study were able to demonstrate acquired skills and stated to appreciate the provided concepts and tools in a way that they want to utilize them in their future endeavors.

This chapter summarizes contributions made in this research work and discusses research hypotheses formulated at its start. An outlook of intended future work to refine and further evaluate the *Integrated Approach* is presented.

6.1 CONTRIBUTIONS

The main contribution of this dissertation is a novel *Integrated Approach* to introducing students into the field of SE methods. This approach considers the high number of existing SE processes/methods as well as the demanded orientation towards an utilization of flexible and composable SE practices instead of rather monolithic SE processes to provide highly transferable knowledge. With its strong foundations in learning theories and based on the SEMAT Essence specification, DGBL, and simulation this approach aims at providing students with competencies—related knowledge, skills, and attitudes—to holistically control an SE endeavor and appreciate the orientation and guidance provided by SE concepts. To provide necessary experiences to develop a professional SE attitude, this approach was explicitly designed to support students in their course and capstone project work that is demanded by all current curriculum guidelines.

To support the phases of the *Integrated Approach* a number of supporting tools, integrating the single phases of the approach, were developed and evaluated:

1. *The Essence Kernel Puzzler*: an explicitly low-threshold offer to get familiar with basic concepts and vocab of the SEMAT Essence kernel.
2. *The Simulation Game*: a simulation-based, intrinsic, tightly linked, and reflective learning game. It integrates the essential concepts of the Essence kernel, including the dynamic steering of an SE endeavor, deep into the gameplay and enables students to experience those concepts in a hazard-free challenging environment to prepare them for their real project work. With its combination of a single player game experience and a team-based collaboration approach, this game requires the cognitive effort of each single player and fosters reflection as well as social interaction to facilitate learning.
3. *A new simulation approach supporting DGBL in SE (method) education* to drive the *Simulation Game*. This approach is based on SEMAT Essence and the DEVS simulation formalism. It was designed to provide deterministic, interactive and highly transparent simulation models focused on essential concepts. This simulation approach does not require a modeler to invest high training effort, not of use outside of this environment. The semantic analysis of interdependencies of Alpha States and Checkpoints of the Essence kernel used to create a simulation model may provide support outside of this approach to any Essence author aiming to provide unambiguous Essence kernel or practice models as well.

4. *The Essence Navigator*: a tool to enact an Essence kernel and run an SE endeavor with its support. This tool was designed to provide orientation and guidance in the *Simulation Game* as well as in a real SE endeavor hence integrating virtual experiences in the *Simulation Game* with real project work afterward.

It has to be mentioned that a considerable part of beneficial characteristics provided by the *Integrated Approach* has to be attributed to SEMAT Essence, esp. the Essence kernel, the underlying emerging OMG standard chosen as the foundation of the *Integrated Approach*.

It is the contribution of this research work to make this new OMG standard accessible in an integrated environment supporting SE education with well-matched concepts, tools, and learning games.

The remainder of this chapter summarizes findings made by this research work and provides an outlook of future work motivated by this research.

6.2 SUMMARY OF RESEARCH QUESTIONS/HYPOTHESES

The research work of this dissertation was guided by research questions and hypotheses stated in 1.4 on page 9. This section summarizes findings based on posed hypotheses.

Starting point of the research was an analysis of existing DGBL approaches in SE education, esp. with regards to software processes and their management.

- *Hypothesis-1*: Constructivist learning approaches, esp. those of social constructivism, are not utilized to their full potential in existing DGBL approaches in SE education.

The analysis revealed that existing approaches were designed with a variety of constructivist educational approaches, like *Active Learning*, *Situated Learning*, and *Learning by Doing*, in mind. As results showed, existing approaches, in general, were supporting to reinforce knowledge learned earlier but seemed inadequate to learn new contents.[126, 301] Most approaches provided single player games, and some tended towards minimum guided instruction approaches. With sole exceptions, none of the approaches integrated scaffolding learning material, e.g. descriptions of the process, or single elements of it, into the learning games.

Considering learning theory and educational approaches, like Vygotsky's *zone of proximal development*, Sweller's *Cognitive Load Theory*, and Reigeluth's *Elaboration Theory*¹, this is unlikely to contribute to

¹ cf. section 2.2.3 on page 24 and section 2.2.5.1 on page 31

ideal learning experiences. Approaches of social constructivism, like collaboration to foster social interaction, discussion, and reflection, are clearly underemployed. With an underemployment of social constructivist approaches, existing solutions indicate not to utilize constructivist learning theory and educational approaches mentioned above to their full potential.

- *Hypothesis-2*: SEMAT Essence facilitates the achievement of stated learning objectives.

SEMAT Essence was chosen as the foundation of the *Integrated Approach* introduced in this thesis. Investigations, including the mapping of demanded learning objectives and outcomes of curriculum guidelines, results of first deployments, and analysis of its support provided to the application of learning theories and educational approaches indicate that SEMAT Essence is facilitating the achievement of stated learning objectives.

- *Hypothesis-3*: An educational simulation model as well as a highly intrinsic digital learning game can be built on top of SEMAT Essence.

The simulation approach, the *Essence Kernel Puzzler*, and the *Simulation Game* prove that a simulation model, as well as different types of learning games, can be built on top of SEMAT Essence, esp. based on its kernel. This includes an intrinsic, tightly coupled, and reflective game that integrates Essence's concepts deeply into the gameplay.

- *Hypothesis-4*: Students, who experience the support of SE methods and tools providing orientation and guidance, appreciate that and develop an attitude wanting to utilize it in future projects.

Results of the questionnaire finishing the conducted case study indicate that participants want to utilize both the SEMAT Essence approach itself as well as the *Essence Navigator*, providing a web-based tool to ease its use, in their future projects.

- *Hypothesis-5*: Preparational activities provided to students can decrease the cognitive load they are facing in their course project work.

The *Integrated Approach* was applied in a case study. Participants played the *Simulation Game* and profited noticeably from their experiences in the following project work. They already had basic knowledge and skills needed. From a *cognitive load theory* perspective, they were able to draw from already structured knowledge in long-term memory. This indicates that the cognitive load students face in their practical project work can be decreased through their appropriate preparation.

- *Hypothesis-6*: Students can be provided with tools to support their a holistic view on their course/capstone project right from its start.

Participants in the case study were able to utilize already familiar Essence concepts and the provided *Essence Navigator* to holistically assess the current state of their endeavor, to use them to guide team discussions, and to steer their endeavor. They did not need considerable time to become acquainted with concepts because they were already familiar with them. This indicates that students can be provided with tools to support a holistic view on their SE endeavor right from its start.

- *Hypothesis-7*: An approach, integrating DGBL activities deeply into an SE course/curriculum, fosters students' competencies with regards to the stated learning objectives.

Furthermore, they were able to identify blind spots of their project work in the course caused by its organizational requirements and to map activities of the given SE process to Activity Spaces of the Essence kernel.

Together with already stated characteristics this indicates that the *Integrated Approach* provided *knowledge, skills, and attitudes*—hence *competencies*²—needed to accomplish these results.

In the case study, a heavily customized version combining *AgileUP* and *OpenUP* to meet organizational requirements of the course was utilized. This given software process was not defined in the Essence language. Its combination with the Essence kernel was born out of the necessity to provide practice guidance to inexperienced students as well as to provide support to keep a holistic perspective on the SE endeavor. A definition of the given software process in the Essence language would have been preferred since it may have provided a more integrated learning experience. Currently, the number of available predefined Essence practices, based on the Essence kernel, is yet limited compared to other industrial SE process standards, e.g. SPEM and its variants³. Hence the definition of own practices and their composition to Essence methods is associated with rather high own efforts. This will change significantly when more practices, as well as guidance to design them, will be provided by and to the Essence community⁴. However, as results of the case study indicate, this combination of a traditionally described software process and the Essence kernel was less obstructive as initially expected and provided students with insights of both worlds.

² cf. section 2.3.2 on page 40

³ cf. section 2.6.2 on page 91

⁴ Recently Park et al.[198] provided such an effort to map the popular agile *Scrum* to Essence.

6.3 FUTURE WORK

After summarizing findings with regards to initially asked research questions and posed hypotheses, this section provides recommendations for future work and research in this field.

The case study conducted to evaluate the developed *Integrated Approach* provided first insights and encouraging results. Since the number of participants was rather small, results cannot be considered to be conclusive. A wider utilization of the proposed *Integrated Approach* is intended to compare the results with those of the case study and to collect a broader base for evaluation of the approach. Providing the SE education community with the encouraging results of the conducted case study through contributions in proper academic channels may arouse the intended interest. With adequate groups of participants, this *Integrated Approach* could be compared in experiments to assess its efficiency. In this dissertation, such experiments were omitted for the reasons given.

After playing the game, a number of students asked for some tutorial or help system added to the game. At running the case study, the author was available on site and questions or technical problems were answered quickly. To support a wide usage of the Simulation Game, some introductory tutorial will be part of future work.

The case study combined a traditionally described software process with the Essence kernel to support students at their course projects. As results indicate, this approach worked out well. It would be interesting to provide the characteristics of the same process defined in Essence practices and composed to an Essence method to students to compare results of that approach to those of the case study.

To provide the *Simulation Game* with the same composed Essence method, the simulation model would have to be extended to support more elements of the Essence language. But even with an integrated Essence method utilized in the real project work, a *Simulation Game* based solely on the Essence kernel may get used to preparing students for their real project work. This likely would increase the cognitive load of students facing elements like sub-Alphas, WorkProducts, etc. for the first time in their course project. It would be interesting to research, if and how much this additional cognitive load impairs students' performance.

Team dynamics identified in the gameplay provide an interesting field for future research. The concepts provided in the *Simulation Game* to foster and trigger social interaction in teams may get opti-

mized based on new findings. It would be interesting to identify different types of players in the game to support them accordingly. The *Simulation Game* provides a novel approach that may be examined in digital learning games of other domains too.

To support the *Integrated Approach*, a number of tools were developed. The priority at their implementation within the scope of this dissertation was to support the case study to evaluate the ideas and concepts integrated into the approach. To make those tools available to a wider audience, some extra work will be needed. This includes the administration of the course environment, implemented to enable a deployment of simulations and game instances for course groups, and the thorough accomplishment of a scalable environment supporting simultaneous working of larger course groups.

It is intended to extend the features of the *Essence Navigator* to provide support of non-kernel Essence language elements. The integration with other standard tools of SE teams, like OSS ticket- and version control systems, is of particular interest since this would provide the opportunity to integrate elements like sub-Alphas and WorkProducts in an efficient way and hence improve the experience of using Essence in real world SE endeavors.

BIBLIOGRAPHY

- [1] Cheryl L. Aasheim and Susan R. Williams. "Knowledge and Skill Requirements for Entry-Level Information Technology Workers: Do Employers in the IT Industry View These Differently than Employers in Other Industries?" In: *Information Technology Faculty Publications*, Paper 1 (2009). URL: <https://digitalcommons.georgiasouthern.edu/information-tech-facpubs/1> (visited on 08/28/2016).
- [2] Tarek Abdel-Hamid. *Software Project Dynamics: An Integrated Approach*. 2nd ed. Prentice Hall, May 1991.
- [3] Tarek K. Abdel-Hamid. "The dynamics of software development project management: An integrative system dynamics perspective." PhD thesis. Massachusetts Institute of Technology, 1984. URL: <http://dspace.mit.edu/handle/1721.1/38235> (visited on 09/16/2016).
- [4] Alain Abran, James W. Moore, Pierre Bourque, Robert Dupuis, and Leonard L. Tripp. *Guide to the Software Engineering Body of Knowledge: 2004 Version SWEBOK*. IEEE Computer Society, Professional Practices Committee, 2004. URL: <http://www.computer.org/portal/web/swebok/purchase>.
- [5] Clark C. Abt. *Serious Games*. en. New York: Viking Press, 1971.
- [6] Academy of Program/Project & Engineering Leadership (APPEL). *National Aeronautics and Space Administration (NASA) Project Management and Systems Engineering Competency Model*. URL: <http://appel.nasa.gov/competency-model/> (visited on 08/26/2016).
- [7] Silvia T. Acuna and Maria I. Sanchez-Segura. *New Trends in Software Process Modelling*. World Scientific Publishing, May 2006.
- [8] M. O. Ahmad, J. Markkula, and M. Oivo. "Kanban in software development: A systematic literature review." In: *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*. Sept. 2013, pp. 9–16. DOI: 10.1109/SEAA.2013.28.
- [9] R. Ahmed, T. Hall, P. Wernick, S. Robinson, and M. Shah. "Software process simulation modelling: A survey of practice." In: *Journal of simulation* 2.2 (2008), pp. 91–102.
- [10] Louis Alfieri, Patricia J. Brooks, Naomi J. Aldrich, and Harriet R. Tenenbaum. "Does discovery-based instruction enhance learning?" In: *Journal of Educational Psychology* 103.1 (2011), pp. 1–18.

- [11] Nauman Bin Ali, Kai Petersen, and Claes Wohlin. "A systematic literature review on the industrial use of software process simulation." In: *Journal of Systems and Software* 97 (Nov. 2014), pp. 65–85. DOI: 10.1016/j.jss.2014.06.059.
- [12] Lorin W. Anderson and David Krathwohl. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. New York: Addison Wesley Longman, 2001.
- [13] Lorin W. Anderson, David R. Krathwohl, and Benjamin Samuel Bloom. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Allyn & Bacon, 2001.
- [14] Angela Burgess, Anne Marie Kelly, Evan M. Butterfield, John Keppler, Dorian McClenahan, Kate Guillemette, and Michelle Phon. *Software Engineering Competency Model (SWECOM) Version 1.0*. IEEE Computer Society, 2014.
- [15] Alex Baker, Emily Oh Navarro, and André Van Der Hoek. "An experimental card game for teaching software engineering." In: *Software Engineering Education and Training, 2003.(CSEE&T 2003). Proceedings. 16th Conference on*. IEEE, 2003, pp. 216–223.
- [16] Alex Baker, Emily Oh Navarro, and André Van Der Hoek. "An experimental card game for teaching software engineering processes." In: *Journal of Systems and Software* 75.1 (2005), pp. 3–16. URL: <http://www.sciencedirect.com/science/article/pii/S0164121204000378> (visited on 09/09/2016).
- [17] Albert Bandura. "Self-efficacy: Toward a unifying theory of behavioral change." English. In: *Psychological Review* 84.2 (1977), pp. 191–215. DOI: 10.1037/0033-295X.84.2.191.
- [18] M. O Barros, A. R Dantas, G. O Veronese, and C. M. L Werner. "Model-driven game development: experience and model enhancements in software project management education." In: *Software Process: Improvement and Practice* 11.4 (2006), pp. 411–421.
- [19] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, and R. Jeffries. "The agile manifesto." In: *The Agile Alliance* (2001).
- [20] Mordechai Ben-Ari. "Constructivism in Computer Science Education." In: *Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '98. New York, NY, USA: ACM, 1998, pp. 257–261. DOI: 10.1145/273133.274308.
- [21] Mordechai Ben-Ari. "Constructivism in computer science education." In: *Journal of Computers in Mathematics and Science Teaching* 20.1 (2001), pp. 45–74.

- [22] Bent Flyvbjerg and Alexander Budzier. "Why Your IT Project May Be Riskier Than You Think." In: *Harvard Business Review* September 2011 (Sept. 2011), pp. 23–25.
- [23] *Berufliche Weiterbildung - Nutzung durch Unternehmen in Deutschland* | Statistik. URL: <http://de.statista.com/statistik/daten/studie/235376/umfrage/nutzung-von-m-learning-durch-unternehmen-in-deutschland-nach-anwendungen/> (visited on 09/08/2016).
- [24] Stefan Biffl, Aybüke Aurum, Barry Boehm, Hakan Erdogmus, and Paul Grünbacher. *Value-Based Software Engineering*. 1st ed. Springer, Berlin, Nov. 2005.
- [25] Benjamin Samuel Bloom, Engelhart, M.D., Furst, E.J., Hill, W.H., and Krathwohl, D.R. "Taxonomy of Educational Objectives: The Classification of Educational Goals." en. In: *Taxonomy of Educational Objectives: The Classification of Educational Goals*. New York: David McKay, 1956.
- [26] George M. Bodner. "Constructivism: A theory of knowledge." In: *Journal of Chemical Education* 63.10 (1986), pp. 873–878.
- [27] Barry Boehm. "A View of 20th and 21st Century Software Engineering." In: *Proceedings of the 28th International Conference on Software Engineering*. ICSE '06. New York, NY, USA: ACM, 2006, pp. 12–29. DOI: 10.1145/1134285.1134288.
- [28] Barry Boehm. "General Theories of Software Engineering (GTSE): Key Criteria and an Example: GTSE 2015 Keynote Address Summary." In: *Proceedings of the Fourth SEMAT Workshop on General Theory of Software Engineering*. GTSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 1–2.
- [29] Barry W. Boehm. "Value-based software engineering: Overview and agenda." In: *Value-based software engineering*. Springer, 2006, pp. 3–14.
- [30] Barry W. Boehm and Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. en. Addison-Wesley Professional, 2004.
- [31] Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby. "Cost models for future software life cycle processes: COCOMO 2.0." In: *Annals of software engineering* 1.1 (1995), pp. 57–94.
- [32] Barry Boehm and Richard Turner. "Balancing Agility and Discipline: A Guide for the Perplexed." Englisch. In: *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*. Edinburgh, Scotland, UK, 2004.
- [33] Derek Bok. *Our Underachieving Colleges: A Candid Look at How Much Students Learn and Why They Should Be Learning More*. en. Princeton University Press, Feb. 2009.

- [34] A. Bollin, E. Hochmüller, and R. T. Mittermeir. "Teaching software project management using simulations." In: *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEET)*. May 2011, pp. 81–90. DOI: 10.1109/CSEET.2011.5876160.
- [35] A. Bollin, E. Hochmüller, R. Mittermeir, and L. Samuelis. "Experiences with Integrating Simulation into a Software Engineering Curriculum." In: *2012 IEEE 25th Conference on Software Engineering Education and Training*. Apr. 2012, pp. 62–71. DOI: 10.1109/CSEET.2012.18.
- [36] Axel Böttcher, Veronika Thurner, and Gerhard Müller. "Kompetenzorientierte Lehre im Software Engineering." In: *SEUH*. 2011, pp. 33–39. URL: <http://ceur-ws.org/Vol-695/beitrag7-boettcher-thurner-mueller.pdf> (visited on 08/26/2016).
- [37] L. Botturi and C. S Loh. "Once Upon a Game: Rediscovering the Roots of Games in Education." In: *Games: purpose and potential in education* (2008). Ed. by Christopher Thomas Miller, pp. 1–22.
- [38] P. Bourque, L. Buglione, A. Abran, and A. April. "Bloom's Taxonomy Levels for Three Software Engineer Profiles." In: *Software Technology and Engineering Practice, International Workshop on*. Los Alamitos, CA, USA: IEEE Computer Society, 2003, pp. 123–129. DOI: <http://doi.ieeecomputersociety.org/10.1109/STEP.2003.6>.
- [39] John D. Bransford, Robert D. Sherwood, Ted S. Hasselbring, Charles K. Kinzer, and Susan M. Williams. "Anchored instruction: Why we need it and how technology can help." In: *Cognition, education, and multimedia: Exploring ideas in high technology* (1990), pp. 115–141.
- [40] J. S Breuer and G. Bente. "Why so serious? On the relation of serious games and learning." In: *Eludamos. Journal for Computer Game Culture* 4.1 (2010), pp. 7–24.
- [41] Johannes Breuer. *Spielend lernen? Eine Bestandsaufnahme zum (Digital) Game-Based Learning*. Dec. 2010. URL: <http://www.lfm-nrw.de/fileadmin/lfm-nrw/Publikationen-Download/Doku41-Spielend-Lernen.pdf> (visited on 08/15/2011).
- [42] Brian Elvesæter, Gorka Benguria, and Sylvia Ilieva. "A Comparison of the Essence 1.0 and SPEM 2.0 Specifications for Software Engineering Methods." In: *Proceedings of the Third Workshop on Process-Based Approaches for Model-Driven Engineering*. PMDE '13. New York, NY, USA: ACM, 2013, 2:1–2:10. DOI: 10.1145/2489833.2489835.

- [43] Kathrin Bröker, Uwe Kastens, and Johannes Magenheim. "Competences of Undergraduate Computer Science Students." In: *KEYCIT 2014: key competencies in informatics and ICT 7* (2015), pp. 77–96.
- [44] Frederick P. Brooks Jr. "NO SILVER BULLET ESSENCE AND ACCIDENTS OF SOFTWARE ENGINEERING." In: *Computer* 20.4 (1986), pp. 10–19. DOI: 10.1109/MC.1987.1663532.
- [45] John Seely Brown, Allan Collins, and Paul Duguid. "Situated cognition and the culture of learning." In: *Educational researcher* 18.1 (1989), pp. 32–42.
- [46] Jerome S. Bruner. "The act of discovery." In: *Harvard educational review* 31 (1961), pp. 21–32.
- [47] Jerome S. Bruner. *The Process of Education*. Harvard University Press, 1977.
- [48] Bundesministerium für Bildung und Forschung. *Arbeiten – Lernen – Kompetenzen entwickeln: Innovationsfähigkeit in einer modernen Arbeitswelt*. Bonn, Berlin: BMBF, 2007.
- [49] Craig Caulfield. "Shall we play a game?" English. PhD thesis. Joondalup, Australia: Edith Cowan University, Nov. 2011. URL: <http://ro.ecu.edu.au/theses/447/>.
- [50] Craig Caulfield, David Veal, and Stanislaw Paul Maj. "Teaching software engineering project management—a novel approach for software engineering programs." In: *Modern Applied Science* 5.5 (2011), pp. 87–104.
- [51] Humberto Cervantes, Serge Haziyeu, Olha Hrytsay, and Rick Kazman. "Smart Decisions: An Architectural Design Game." In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ICSE '16. New York, NY, USA: ACM, 2016, pp. 327–335. DOI: 10.1145/2889160.2889184.
- [52] R. O. Chaves, C. G. von Wangenheim, J. C. C. Furtado, S. R. B. Oliveira, A. Santos, and E. L. Favero. "Experimental Evaluation of a Serious Game for Teaching Software Process Modeling." In: *IEEE Transactions on Education* 58.4 (Nov. 2015), pp. 289–296. DOI: 10.1109/TE.2015.2411573.
- [53] Jenova Chen. "Flow in games (and everything else)." In: *Communications of the ACM* 50.4 (2007), pp. 31–34.
- [54] KeungSik Choi, Doo-Hwan Bae, and TagGon Kim. "An approach to a hybrid software process simulation using the DEVS formalism." en. In: *Software Process: Improvement and Practice* 11.4 (July 2006), pp. 373–383. DOI: 10.1002/spip.284.
- [55] Noam Chomsky. "A review of BF Skinner's Verbal Behavior." In: *Language* 35.1 (1959), pp. 26–58.

- [56] Alex Chung Hen Chow and Bernard P. Zeigler. "Parallel DEVS: A Parallel, Hierarchical, Modular, Modeling Formalism." In: *Proceedings of the 26th Conference on Winter Simulation*. WSC '94. San Diego, CA, USA: Society for Computer Simulation International, 1994, pp. 716–722.
- [57] Ioannis Christou, Stavros Ponis, and Eleni Palaiologou. "Using the agile unified process in banking." In: *IEEE Software* 27.3 (2010), pp. 72–79.
- [58] Committee on Developments in the Science of Learning with additional material from the Committee on Learning Research and Educational Practice and National Research Council. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition*. 2nd ed. National Academies Press, Sept. 20.
- [59] *Computerspiele in Deutschland - Spieldauer pro Woche 2011* | Statistik. URL: <http://de.statista.com/statistik/daten/studie/168810/umfrage/spieldauer-von-computerspielen-pro-woche/> (visited on 08/17/2016).
- [60] Thomas M. Connolly and Carolyn E. Begg. "A Constructivist-Based Approach to Teaching Database Analysis and Design." In: *Journal of Information Systems Education* (2005), pp. 43–53.
- [61] Thomas M. Connolly, Elizabeth A. Boyle, Ewan MacArthur, Thomas Hainey, and James M. Boyle. "A systematic literature review of empirical evidence on computer games and serious games." In: *Computers & Education* 59.2 (Sept. 2012), pp. 661–686. DOI: 10.1016/j.compedu.2012.03.004.
- [62] Thomas M. Connolly, Mark Stansfield, and Thomas Hainey. "An application of games-based learning within software engineering." en. In: *British Journal of Educational Technology* 38.3 (May 2007), pp. 416–428. DOI: 10.1111/j.1467-8535.2007.00706.x.
- [63] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. 3rd Edition. Cambridge: MIT press, 2009. (Visited on 07/11/2016).
- [64] Mihaly Csikszentmihalyi. *Flow: the psychology of optimal experience*. New York, N.Y.: Harper Perennial, 1991.
- [65] D. Cunningham and T. Duffy. "Constructivism: Implications for the design and delivery of instruction." In: *Handbook of research for educational communications and technology* (1996), pp. 170–198.
- [66] Damien Djaouti, Julian Alvarez, Jean-Pierre Jessel, and Gilles Méthel. "PLAY, GAME, WORLD: ANATOMY OF A VIDEOGAME." In: *International Journal of Intelligent Games & Simulation* 5.1 (Jan. 2008).

- [67] Alexandre R. Dantas, Márcio de Oliveira Barros, and Cláudia Maria Lima Werner. "A Simulation-Based Game for Project Management Experiential Learning." In: *SEKE*. Vol. 19. 2004, p. 24.
- [68] Chris Dede. "The evolution of constructivist learning environments: Immersion in distributed, virtual worlds." In: *Educational technology* 35.5 (1995), pp. 46–52.
- [69] Marcus Deininger and Kurt Schneider. "Teaching software project management by simulation—Experiences with a comprehensive model." In: *Conference on Software Engineering Education*. Springer, 1994, pp. 227–242.
- [70] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. "From Game Design Elements to Gamefulness: Defining "Gamification"." In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. MindTrek '11. New York, NY, USA: ACM, 2011, pp. 9–15. DOI: 10.1145/2181037.2181040.
- [71] Christoph Dickmann, Harald Klein, Thomas Birkhölzer, Wolfgang Fietz, Jürgen Vaupel, and Ludger Meyer. "Deriving a Valid Process Simulation from Real World Experiences." en. In: *Software Process Dynamics and Agility*. Ed. by Qing Wang, Dietmar Pfahl, and David M. Raffo. Lecture Notes in Computer Science 4470. DOI: 10.1007/978-3-540-72426-1_23. Springer Berlin Heidelberg, May 2007, pp. 272–282.
- [72] Oxford Dictionaries. *Oxford Dictionary of English, 2nd Edition*. Englisch. Ed. by Catherine Soanes and Angus Stevenson. 2nd ed. Oxford University Press, 2010.
- [73] Damien Djaouti, Julian Alvarez, Jean-Pierre Jessel, and Olivier Rampnoux. "Origins of Serious Games." In: *Serious Games and Edutainment Applications*. Ed. by Minhua Ma, Andreas Oikonomou, and Lakhmi C. Jain. Springer London, 2011, pp. 25–43. DOI: 10.1007/978-1-4471-2161-9_3.
- [74] Claudio Dondi and Michela Moretti. "A methodological proposal for learning games selection and quality assessment." In: *British Journal of Educational Technology* 38.3 (2007), pp. 502–512.
- [75] Paolo Donzelli and Giuseppe Iazeolla. "A Hybrid Software Process Simulation Model." In: *SOFTWARE PROCESS – IMPROVEMENT AND PRACTICE* 6 (2001), pp. 97–110.
- [76] A Drappa and J Ludewig. "Quantitative modeling for the interactive simulation of software projects." In: *Journal of Systems and Software* 46.2–3 (Apr. 1999), pp. 113–122. DOI: 10.1016/S0164-1212(99)00005-9.

- [77] Anke Drappa and Jochen Ludewig. "Simulation in software engineering training." In: *Proceedings of the 22nd international conference on Software engineering*. ACM, 2000, pp. 199–208.
- [78] Marcy Perkins Driscoll and Marcy P. Driscoll. "Gagné's Theory of Instruction." In: *Psychology of learning for instruction (2nd ed.)* Boston: Allyn & Bacon, 2000, pp. 341–372.
- [79] Charles Edeki. "Agile Unified Process." In: *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE* 1.3 (Sept. 2013), pp. 13–17.
- [80] Khaled El Emam and A. Gunes Koru. "A replicated survey of IT software project failures." In: *Software, IEEE* 25.5 (2008), pp. 84–90.
- [81] Employment and Training Administration, United States Department of Labor. *Information Technology Competency Model*. Sept. 2012. URL: <http://www.careeronestop.org/competency%20model/competency-models/pyramid-download.aspx?industry=%20information-technology> (visited on 08/15/2016).
- [82] Entertainment Software Association. *2016 Sales, Demographic and Usage Data: Essential Facts About The Computer And Video Game Industry*. 2016. URL: <http://essentialfacts.theesa.com/> (visited on 09/08/2016).
- [83] Michael D. Ernst and John Chapin. "The groupthink specification exercise." In: *Proceedings of the 27th international conference on Software engineering*. ICSE '05. New York, NY, USA: ACM, 2005, pp. 617–618. DOI: 10.1145/1062455.1062568.
- [84] J.L. Eveleens and C. Verhoef. "The rise and fall of the Chaos report figures." In: *IEEE Software* 27.1 (Jan. 2010), pp. 30–36. DOI: 10.1109/MS.2009.154.
- [85] Facebook Inc. *A JavaScript library for building user interfaces | React*. URL: <https://facebook.github.io/react/> (visited on 08/13/2016).
- [86] Nuno H. Flores, Ana C. R. Paiva, and Pedro Letra. "Software Engineering Management Education through Game Design Patterns." In: *Procedia - Social and Behavioral Sciences*. 2nd International Conference on Higher Education Advances, HEAd'16, 21–23 June 2016, València, Spain 228 (July 2016), pp. 436–442. DOI: 10.1016/j.sbspro.2016.07.067.
- [87] Sara de Freitas and Fotis Liarokapis. "Serious Games: A New Paradigm for Education?" In: *Serious Games and Edutainment Applications*. Springer, 2011, pp. 9–23.
- [88] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, and Grady Booch. *Design Patterns: Elements of Reusable Object-Oriented Software*. Englisch. 1st ed. Addison-Wesley Professional, 1994.

- [89] L. Ganesh. "Board Game as a Tool to Teach Software Engineering Concept – Technical Debt." In: *2014 IEEE Sixth International Conference on Technology for Education (T4E)*. Dec. 2014, pp. 44–47. DOI: 10.1109/T4E.2014.28.
- [90] Ángel Garcia-Crespo, Ricardo Colomo-Palacios, Juan Miguel Gómez-Berbís, and Fernando Paniagua-Martín. "A Case of System Dynamics Education in Software Engineering Courses." In: *IEEE MULTIDISCIPLINARY ENGINEERING EDUCATION MAGAZINE* 3.2 (July 2008), pp. 52–59.
- [91] Gartner, Inc. *Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013*. Feb. 2014. URL: <http://www.gartner.com/newsroom/id/2665715> (visited on 02/20/2014).
- [92] David C. Geary. "An Evolutionary Perspective on Learning Disability in Mathematics." In: *Developmental Neuropsychology* 32.1 (July 2007), pp. 471–519. DOI: 10.1080/87565640701360924.
- [93] James Paul Gee. *What Video Games Have to Teach Us About Learning and Literacy*. Palgrave Macmillan, 2007.
- [94] Georg Hagel and Jürgen Mottok. "Planspiel und Briefmethode für die Software Engineering Ausbildung - ein Erfahrungsbericht." In: *Tagungsband des 12. Workshops "Software Engineering im Unterricht der Hochschulen" 2011*. Ed. by Jochen Ludewig and Axel Böttcher. Vol. Vol-695. München: CEUR Workshop Proceedings (CEUR-WS.org), Feb. 2011, pp. 10–15. URL: <http://ceur-ws.org/Vol-695/beitrag3-hagel-mottok.pdf> (visited on 10/16/2016).
- [95] Gesellschaft für Informatik e.V. (GI). *Empfehlungen für Bachelor- und Masterprogramme im Studienfach Informatik an Hochschulen*. July 2016.
- [96] Robert L. Glass. "The Standish report: does it really describe a software crisis?" In: *Communications of the ACM* 49.8 (2006), pp. 15–16.
- [97] Carolin Gold-Veerkamp. *Erhebung von Soll-Kompetenzen im Software Engineering*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015.
- [98] Daniel Graziotin and Pekka Abrahamsson. "A Web-based modeling tool for the SEMAT Essence theory of software engineering." In: *Journal of Open Research Software* 1.1 (Sept. 2013). DOI: 10.5334/jors.ad.
- [99] Christopher Guindon. *Eclipse Process Framework Project*. und. Jan. 2013. URL: <http://projects.eclipse.org/projects/technology.epf> (visited on 09/15/2016).

- [100] Christopher Guindon. *Eclipse Process Framework Project*. en. Text. Jan. 2013. URL: <https://projects.eclipse.org/projects/technology.epf> (visited on 08/31/2016).
- [101] Bjorn Gustafsson. "OpenUP – the best of two worlds." In: *Methods & Tools* 16.1 (2008), pp. 21–32.
- [102] S. Hadjerrouit. "Learner-centered web-based instruction in software engineering." In: *IEEE Transactions on Education* 48.1 (Feb. 2005), pp. 99–104. DOI: 10.1109/TE.2004.832871.
- [103] Said Hadjerrouit. "Constructivism As Guiding Philosophy for Software Engineering Education." In: *SIGCSE Bull.* 37.4 (Dec. 2005), pp. 45–49. DOI: 10.1145/1113847.1113875.
- [104] Thomas Hainey. "Using games-based learning to teach requirements collection and analysis at tertiary education level." PhD Thesis. University of the West of Scotland, 2010. URL: a.
- [105] Thomas Hainey, Thomas M. Connolly, Mark Stansfield, and Elizabeth A. Boyle. "Evaluation of a game to teach requirements collection and analysis in software engineering at tertiary education level." In: *Computers & Education* 56.1 (2011), pp. 21–35.
- [106] Jon G. Hall and Lucia Rapanotti. "Towards a Design-theoretic Characterisation of Software Development Process Models." In: *Proceedings of the Fourth SEMAT Workshop on General Theory of Software Engineering*. GTSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 3–14.
- [107] Linda Harasim. *Learning Theory and Online Technologies*. English. New York, NY: Routledge, Aug. 2011.
- [108] Ville T. Heikkilä, Maria Paasivaara, and Casper Lassenius. "Teaching University Students Kanban with a Collaborative Board Game." In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ICSE '16. New York, NY, USA: ACM, 2016, pp. 471–480. DOI: 10.1145/2889160.2889201.
- [109] Alex Hern. "Did your Adobe password leak? Now you and 150m others can check." en-GB. In: *The Guardian* (Nov. 2013). URL: <http://www.theguardian.com/technology/2013/nov/07/adobe-password-leak-can-check> (visited on 02/21/2014).
- [110] Nien-Lin Hsueh, Wen-Hsiang Shen, Zhi-Wei Yang, and Don-Lin Yang. "Applying UML and software simulation for process definition, verification, and validation." In: *Information and Software Technology* 50.9 (2008), pp. 897–911.
- [111] Johan Huizinga. *Homo Ludens: A Study of The Play-Element In Culture*. 22nd ed. London: Routledge & Kegan Paul, 1949.

- [112] IEEE Computer Society. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. Ed. by Pierre Bourque and Richard E. Fairley. 3rd. Los Alamitos, CA, USA: IEEE Computer Society Press, 2014.
- [113] International Organization for Standardisation (ISO). *ISO/IEC 24744:2007 - Software Engineering – Metamodel for Development Methodologies*. Feb. 2007. URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=38854 (visited on 10/31/2014).
- [114] ISO/IEC. *ISO/IEC 24744:2014 - Software engineering – Metamodel for development methodologies*. Nov. 2014. URL: http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=62644 (visited on 08/31/2016).
- [115] Ivar Jacobson International. *Developing and Customizing Practices | EssWork Practice Workbench*. 2014. URL: http://www.ivarjacobson.com/EssWork_Practice_Workbench/ (visited on 06/25/2014).
- [116] Ivar Jacobson, Bertrand Meyer, and Richard Soley. *The SEMAT Initiative: A Call for Action*. Sept. 2009. URL: <http://www.drdobbs.com/architecture-and-design/the-semat-initiative-a-call-for-action/222001342> (visited on 09/03/2016).
- [117] Ivar Jacobson, Bertrand Meyer, and Richard Soley. *SOFTWARE ENGINEERING METHOD AND THEORY – A VISION STATEMENT*. Jan. 2010. URL: <http://semat.org/documents/20181/27952/SEMAT-vision.pdf/16059a36-a0ba-4405-b883-4a11a2%20131cea>.
- [118] Ivar Jacobson, Pan-Wei Ng, Paul E. McMahon, Ian Spence, and Svante Lidman. *The Essence of Software Engineering: Applying the SEMAT Kernel*. English. 1 edition. Upper Saddle River, NJ: Addison-Wesley Professional, Jan. 2013.
- [119] Ivar Jacobson, Pan-Wei Ng, Paul McMahon, Ian Spence, and Svante Lidman. "The Essence of Software Engineering: The SEMAT Kernel." In: *Queue* 10.10 (Oct. 2012), 40:40–40:51. DOI: 10.1145/2381996.2389616. URL: <http://doi.acm.org/10.1145/2381996.2389616> (visited on 07/01/2014).
- [120] Ivar Jacobson, Pan-Wei Ng, Ian Spence, and Paul E. McMahon. "Major-league SEMAT: Why Should an Executive Care?" In: *Queue* 12.2 (Feb. 2014), 20:20–20:28. DOI: 10.1145/2578508.2590809. URL: <http://doi.acm.org/10.1145/2578508.2590809> (visited on 10/24/2014).
- [121] Ivar Jacobson, Pan Wei Ng, and Ian Spence. "Enough of Processes: Let's Do Practices Part 2." In: *Dr. Dobb's Journal* 2007.4 (Apr. 2007), pp. 28–32. URL: <http://www.drdobbs.com/0opKZ/architecture-and-design/enough-of-processes-lets-do-practices/198800543> (visited on 01/28/2016).

- [122] Ivar Jacobson, Pan Wei Ng, and Ian Spence. "Enough of Processes: Let's Do Practices Part 1." In: *Dr. Dobb's Journal* 2007.3 (Mar. 2007). URL: <http://www.drdobbs.com/architecture-and-design/enough-of-processes-lets-do-practices-pa/198000264> (visited on 01/28/2016).
- [123] Ivar Jacobson, Pan, Pan Wei Ng, and Ian Spence. "Enough of Processes: Let's Do Practices: Part 3." In: *Dr. Dobb's Journal* 2007.5 (May 2007), pp. 34–38. URL: <http://www.drdobbs.com/%20architecture-and-design/enough-of-processes-lets-do-practices-pa/199204020> (visited on 01/28/2016).
- [124] Ivar Jacobson, Ian Spence, and Pan-Wei Ng. "Agile and SEMAT - Perfect Partners." In: *Queue* 11.9 (Sept. 2013), 30:30–30:41. DOI: 10.1145/2538031.2541674. URL: <http://doi.acm.org/10.1145/2538031.2541674> (visited on 10/24/2014).
- [125] A. Jain and B. Boehm. "SimVBSE: Developing a game for value-based software engineering." In: *Proceedings of the 19th Conference on Software Engineering Education and Training*. 2006, pp. 103–114.
- [126] Shu Jiang, He Zhang, Chao Gao, Dong Shao, and Guoping Rong. "Process Simulation for Software Engineering Education." In: *Proceedings of the 2015 International Conference on Software and System Process*. ICSSP 2015. New York, NY, USA: ACM, 2015, pp. 147–156. DOI: 10.1145/2785592.2785606.
- [127] Pontus Johnson, Michael Goedicke, and Ivar Jacobson. "Proceedings of the 3rd SEMAT Workshop on General Theories of Software Engineering." In: ACM, June 2014. URL: <http://dl.acm.org/citation.cfm?id=2593752> (visited on 08/31/2016).
- [128] Pontus Johnson, Ivar Jacobson, Michael Goedicke, and Mira Kajko-Mattsson. "2Nd SEMAT Workshop on a General Theory of Software Engineering (GTSE 2013)." In: *Proceedings of the 2013 International Conference on Software Engineering*. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 1525–1526. URL: <http://dl.acm.org/citation.cfm?id=2486788>. 2487066 (visited on 08/31/2016).
- [129] Pontus Johnson, Paul Ralph, Michael Goedicke, Pan-Wei Ng, Klaas-Jan Stol, Kari Smolander, Jaakov Exman, and Dewayne E Perry. "Report on the Second SEMAT Workshop on General Theory of Software Engineering (GTSE 2013)." In: *SIGSOFT Softw. Eng. Notes* 38.5 (Aug. 2013), pp. 47–50. DOI: 10.1145/2507288.2529923. URL: <http://doi.acm.org/10.1145/2507288.2529923> (visited on 08/31/2016).
- [130] Steven Johnson. *Everything Bad is Good for You*. Penguin, May 2006.

- [131] Aditya Johri and Barbara M. Olds. "Situated Engineering Learning: Bridging Engineering Education Research and the Learning Sciences." en. In: *Journal of Engineering Education* 100.1 (Jan. 2011), pp. 151–185. DOI: 10.1002/j.2168-9830.2011.tb00007.x.
- [132] Joint Task Force on Computing Curricula IEEE Computer Society Association for Computing Machinery. *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. Aug. 2004.
- [133] Joint Task Force on Computing Curricula IEEE Computer Society Association for Computing Machinery. *Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. Feb. 2015.
- [134] David H. Jonassen. "Thinking Technology: Toward a Constructivist Design Model." en. In: *Educational Technology* 34.4 (1994), pp. 34–37.
- [135] Jöran Pieper. "Alles nur Spielerei? Neue Ansätze für digitales spielbasiertes Lernen von Softwareprozessen." In: *Tagungsband des 13. Workshops "Software Engineering im Unterricht der Hochschulen" 2013*. Ed. by Andreas Spillner and Horst Lichter. Vol. Vol-956. Aachen: CEUR Workshop Proceedings, Feb. 2013, pp. 131–139. URL: http://ceur-ws.org/Vol-956/S5_Paper1.pdf (visited on 03/01/2013).
- [136] Magne Jørgensen and Kjetil Moløkken-Østvold. "How large are software cost overruns? A review of the 1994 CHAOS report." In: *Information and Software Technology* 48.4 (2006), pp. 297–301.
- [137] June Sung Park, Ivar Jacobson, Barry Myburgh, Pontus Johnson, and Paul E. McMahon. *SEMAT Yesterday, Today and Tomorrow: An Industry Perspective*. Feb. 2014. URL: http://semat.org/news/-/asset_publisher/eaHEtyeuE9wP/content/semat-yesterday-today-and-tomorrow (visited on 07/13/2016).
- [138] Katie Salen and Eric Zimmerman. *Rules of Play: Game Design Fundamentals*. Cambridge: MIT Press, 2003.
- [139] Marc I. Kellner, Raymond J. Madachy, and David M. Raffo. "Software process simulation modeling: why? what? how?" In: *Journal of Systems and Software* 46.2 (1999), pp. 91–105.
- [140] Ken Schwaber. *Scrum and The Perfect Storm*. Jan. 2003. URL: <http://www.controlchaos.com/storage/scrum-articles/Scrum%20and%20The%20Perfect%20Storm.pdf> (visited on 08/16/2016).

- [141] Ken Schwaber and Jeff Sutherland. *The Scrum Guide: The Definite Guide to Scrum-The Rules of the Game*. June 2016. URL: <http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf> (visited on 09/02/2016).
- [142] Mark Kennaley. *SDLC 3.0: Beyond a Tacit Understanding of Agile: Towards the Next Generation of Software Engineering*. Fourth Medium Press, Jan. 2010.
- [143] Michael Kerres. *Multimediale und telemediale Lernumgebungen: Konzeption und Entwicklung*. Walter de Gruyter, Jan. 2001.
- [144] P. A Kirschner, J. Sweller, and R. E Clark. "Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching." In: *Educational psychologist* 41.2 (2006), pp. 75–86.
- [145] E. Knauss, K. Schneider, and K. Stapel. "A Game for Taking Requirements Engineering More Seriously." In: *2008 Third International Workshop on Multimedia and Enjoyable Requirements Engineering - Beyond Mere Descriptions and with More Fun and Games*. Sept. 2008, pp. 22–26. DOI: 10.1109/MERE.2008.1.
- [146] Henrik Kniberg. *Scrum and XP from the Trenches*. C4Media, 2007.
- [147] David R. Krathwohl. "A revision of Bloom's taxonomy: An overview." In: *Theory into practice* 41.4 (2002), pp. 212–218.
- [148] Thomas S. Kuhn. *The Structure of Scientific Revolutions*. en. 2nd edition. Chicago, IL: University of Chicago Press, 1970.
- [149] Marco Kuhrmann, Daniel Méndez Fernández, and Ragna Steenweg. "Systematic Software Process Development: Where Do We Stand Today?" In: *Proceedings of the 2013 International Conference on Software and System Process*. ICSSP 2013. New York, NY, USA: ACM, 2013, pp. 166–170. DOI: 10.1145/2486046.2486077.
- [150] Robert Lagerström and Mathias Ekstedt. "Extending a General Theory of Software to Engineering." In: *Proceedings of the 3rd SEMAT Workshop on General Theories of Software Engineering*. GTSE 2014. New York, NY, USA: ACM, 2014, pp. 36–39. DOI: 10.1145/2593752.2593759.
- [151] Phillip A. Laplante. *What Every Engineer Should Know about Software Engineering*. CRC Press, Apr. 2007.
- [152] Timothy C. Lethbridge, Jorge Diaz-Herrera, Richard J. Jr. LeBlanc, and J. Barrie Thompson. "Improving Software Practice Through Education: Challenges and Future Trends." In: *2007 Future of Software Engineering*. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 12–28. DOI: 10.1109/FOSE.2007.13.

- [153] J. E. N. Lino, M. A. Paludo, F. V. Binder, S. Reinehr, and A. Malucelli. "Project management game 2D (PMG-2D): A serious game to assist software project managers training." In: *IEEE Frontiers in Education Conference (FIE)*, 2015. Oct. 2015, pp. 1–8. DOI: 10.1109/FIE.2015.7344168.
- [154] Dapeng Liu, Qing Wang, and Junchao Xiao. "The Role of Software Process Simulation Modeling in Software Risk Management: A Systematic Review." In: *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. ESEM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 302–311. DOI: 10.1109/ESEM.2009.5315982.
- [155] Jochen Ludewig. "Models in software engineering – an introduction." en. In: *Software and Systems Modeling* 2.1 (Feb. 2003), pp. 5–14. DOI: 10.1007/s10270-003-0020-3.
- [156] Jochen Ludewig, Thomas Bassler, Marcus Deininger, Kurt Schneider, and Jürgen Schwille. "SESAM-simulating software projects." In: *Software Engineering and Knowledge Engineering, 1992. Proceedings., Fourth International Conference on*. IEEE, 1992, pp. 608–615.
- [157] Jochen Ludewig and Marcus Deininger. "Teaching software project management by simulation: The SESAM project." In: *5th European Conference on Software Quality, Dublin*. 1996, pp. 417–426.
- [158] Charles M. Macal, Arnold Buss, Susan K. Heath, and Sally C. Brailsford. *Cross-Paradigm Simulation Modeling: Challenges and Successes*. NAVAL POSTGRADUATE SCHOOL MONTEREY CA GRADUATE SCHOOL OF BUSINESS and PUBLIC POLICY, 2011. URL: <http://www.stormingmedia.us/93/9358/A935855.html> (visited on 03/18/2013).
- [159] Raymond J. Madachy. *Software Process Dynamics*. 28th. Wiley-IEEE Press, Jan. 2008.
- [160] P. Mandl-Striegnitz, A. Drappa, and H. Lichter. "Simulating Software Projects—An Approach for Teaching Project Management." In: *Proceedings of the INSPIRE III: Process Improvement Through Training and Education* (1998), pp. 87–98. DOI: 10.1.1.70.6592.
- [161] Patricia Mandl-Striegnitz. "How to successfully use software project simulation for educating software project managers." In: *Frontiers in Education Conference, 2001. 31st Annual*. Vol. 1. IEEE, 2001, T2D–19. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=963884 (visited on 09/09/2016).
- [162] Mark Kennaley. *Let's Build a Smarter Method: SDLC 3.0: A Complex Adaptive System of Patterns*. 2010.

- [163] Robert Martin and David Raffo. "Application of a hybrid process simulation model to a software development project." In: *Journal of Systems and Software*. Software Process Simulation Modeling 59.3 (Dec. 2001), pp. 237–246. DOI: 10.1016/S0164-1212(01)00065-6.
- [164] J. A. McCall, G. Y. Wong, and A. H. Stone. *A Simulation Modeling Approach to Understanding the Software Development Process*. Tech. rep. June 1979.
- [165] Jeroen J. G. van Merriënboer and John Sweller. "Cognitive Load Theory and Complex Learning: Recent Developments and Future Directions." en. In: *Educational Psychology Review* 17.2 (2005), pp. 147–177. DOI: 10.1007/s10648-005-3951-0.
- [166] David Michael and Sande Chen. *Serious Games: Games That Educate, Train, and Inform*. 1st ed. Course Technology PTR, Oct. 2005.
- [167] R. T. Mittermeir, E. Hochmüller, A. Bollin, S. Jäger, and M. Nusser. "AMEISE – A Media Education Initiative for Software Engineering Concepts, the Environment and Initial Experiences." In: *Proceedings of the Interactive Computer aided Learning (ICL) 2003 International Workshop*. Villach, Austria, Sept. 2003.
- [168] Ronald Moen and Clifford Norman. *Evolution of the PDCA cycle*. 2006. URL: http://pkpinc.com/files/NA01_Moen_Norman_fullpaper.pdf (visited on 09/02/2016).
- [169] Ana M. Moreno, Maria-Isabel Sanchez-Segura, Fuensanta Medina-Dominguez, and Laura Carvajal. "Balancing software engineering education and industrial needs." In: *Journal of Systems and Software*. Software Ecosystems 85.7 (July 2012), pp. 1607–1620. DOI: 10.1016/j.jss.2012.01.060.
- [170] Lorenzo Moreno, Carina Gonzalez, Ivan Castilla, Evelio Gonzalez, and Jose Sigut. "Applying a constructivist and collaborative methodological approach in engineering education." In: *Computers & Education* 49.3 (Nov. 2007), pp. 891–915. DOI: 10.1016/j.compedu.2005.12.004.
- [171] A. Nassal. "A general framework for software project management simulation games." In: *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*. June 2014, pp. 1–5. DOI: 10.1109/CISTI.2014.6877074.
- [172] Alexander Nassal. "Projektmanagement spielend lernen." In: *Software Engineering im Unterricht der Hochschulen (SEUH)*. 2015, pp. 53–64.

- [173] Alexander Nassal and Matthias Tichy. "Modeling Human Behavior for Software Engineering Simulation Games." In: *Proceedings of the 5th International Workshop on Games and Software Engineering*. GAS '16. New York, NY, USA: ACM, 2016, pp. 8–14. DOI: 10.1145/2896958.2896961.
- [174] Emily Navarro. "SimSE: A Software Engineering Simulation Environment for Software Process Education." PhD thesis. Irvine, CA: University of California, 2006. URL: <http://www.ics.uci.edu/~emilyo/papers/Dissertation.pdf> (visited on 02/25/2011).
- [175] Emily Oh Navarro, A. Baker, and A. Van Der Hoek. "Teaching software engineering using simulation games." In: *Proceedings of the International Conference on Simulation in Education (IC-SiE'04)*. San Diego, California, Jan. 2004, pp. 9–14.
- [176] Emily Oh Navarro and André van der Hoek. "SIMSE: An Interactive Simulation Game for Software Engineering Education." In: *Proceedings of the 7th IASTED International Conference on Computers and Advanced Technology in Education (CATE)*. Kauai, Hawaii, Aug. 2004, pp. 12–17.
- [177] Emily Oh Navarro and André Van Der Hoek. "Design and evaluation of an educational software process simulation environment and associated model." In: *18th Conference on Software Engineering Education & Training (CSEET'05)*. IEEE, 2005, pp. 25–32.
- [178] Emily Oh Navarro and André Van Der Hoek. "Design and evaluation of an educational software process simulation environment and associated model." In: *Software Engineering Education and Training, 2005. CSEE&T 2005. Proceedings. 18th Conference on*. 2005, pp. 25–32.
- [179] Emily Oh Navarro and André Van Der Hoek. "Software process modeling for an educational software engineering simulation game." In: *Software Process: Improvement and Practice* 10.3 (2005), pp. 311–325. URL: <http://onlinelibrary.wiley.com/doi/10.1002/spip.232/abstract> (visited on 10/29/2016).
- [180] Emily Oh Navarro and André Van Der Hoek. "Software process modeling for an educational software engineering simulation game." In: *Software Process: Improvement and Practice* 10.3 (2005), pp. 311–325.
- [181] Emily Oh Navarro and André Van Der Hoek. "Comprehensive evaluation of an educational software engineering simulation environment." In: *20th Conference on Software Engineering Education & Training (CSEET'07)*. IEEE, 2007, pp. 195–202.

- [182] Emily Oh Navarro and André Van Der Hoek. "Multi-Site Evaluation of SimSE." In: *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*. Vol. 41. 1. ACM. Chattanooga, TN, USA: ACM, Mar. 2009, pp. 326–330. DOI: 10.1145/1508865.1508981.
- [183] Pan-Wei Ng. "Theory Based Software Engineering with the SEMAT Kernel: Preliminary Investigation and Experiences." In: *Proceedings of the 3rd SEMAT Workshop on General Theories of Software Engineering*. GTSE 2014. New York, NY, USA: ACM, 2014, pp. 13–20. DOI: 10.1145/2593752.2593756.
- [184] Pan-Wei Ng and Shihong Huang. "Essence: A framework to help bridge the gap between software engineering education and industry needs." In: *Proceedings of the 26th IEEE Conference on Software Engineering Education and Training (CSEET)*, 2013. San Francisco, CA, USA: IEEE, May 2013, pp. 304–308. DOI: 10.1109/CSEET.2013.6595266.
- [185] J. Noll, A. Butterfield, K. Farrell, T. Mason, M. McGuire, and R. McKinley. "GSD Sim: A Global Software Development Game." In: *2014 IEEE International Conference on Global Software Engineering Workshops*. Aug. 2014, pp. 15–20. DOI: 10.1109/ICGSEW.2014.12.
- [186] Object Management Group. *Software & Systems Process Engineering Meta-Model Specification (SPEM) Version 2.0*. Apr. 2008. URL: <http://www.omg.org/spec/SPEM/2.0/> (visited on 11/13/2012).
- [187] Object Management Group. *Kernel and Language for Software Engineering Methods (Essence) Version 1.0*. Nov. 2014. URL: <http://www.omg.org/spec/Essence/Current> (visited on 11/03/2014).
- [188] Object Management Group. *Kernel and Language for Software Engineering Methods (Essence) Version 1.1*. Dec. 2015. URL: <http://www.omg.org/spec/Essence/1.1/> (visited on 01/23/2016).
- [189] Object Management Group (OMG). *Essence in Practice: A Revolution in Software Engineering? OMG Technical Meeting Special Event*. URL: http://www.omg.org/news/meetings/tc/berlin-15/special-events/Essence_Day.htm (visited on 08/13/2016).
- [190] Object Management Group (OMG). *Unified Modeling Language (UML)*. URL: <http://www.omg.org/spec/UML/> (visited on 07/28/2016).
- [191] Object Management Group (OMG). *Unified Modeling Language™ (UML®), V2.4.1, Infrastructure Specification*. Aug. 2011. URL: <http://www.omg.org/spec/UML/2.4.1/> (visited on 07/28/2016).

- [192] L. Osterweil. "Software Processes Are Software Too." In: *Proceedings of the 9th International Conference on Software Engineering*. ICSE '87. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987, pp. 2–13.
- [193] Leon J. Osterweil. "Unifying microprocess and macroprocess research." In: *Software Process Workshop*. Vol. 3840. LNCS. Springer, 2005, pp. 68–74.
- [194] Fred Paas, Alexander Renkl, and John Sweller. "Cognitive Load Theory: Instructional Implications of the Interaction between Information Structures and Cognitive Architecture." en. In: *Instructional Science* 32.1-2 (Jan. 2004), pp. 1–8. DOI: 10.1023/B:TRUC.0000021806.17516.d0.
- [195] Fred Paas and John Sweller. "Implications of cognitive load theory for multimedia learning." In: *The Cambridge handbook of multimedia learning* 27 (2014), pp. 27–42.
- [196] Seymour Papert. "Does easy do it? Children, games, and learning." In: *Game developer magazine* 5.6 (1988).
- [197] June Sung Park. "Essence-based, Goal-driven Adaptive Software Engineering." In: *Proceedings of the Fourth SEMAT Workshop on General Theory of Software Engineering*. GTSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 33–38. URL: <http://dl.acm.org/citation.cfm?id=2820167.2820176> (visited on 08/31/2016).
- [198] June Sung Park, Paul E. McMahon, and Barry Myburgh. "Scrum Powered by Essence." In: *SIGSOFT Softw. Eng. Notes* 41.1 (Feb. 2016), pp. 1–8. DOI: 10.1145/2853073.2853088.
- [199] Paul E. McMahon. *Essence: Why do we need it?* Nov. 2013. URL: <http://sematblog.wordpress.com/2013/11/16/essence-why-do-we-need-it/> (visited on 10/14/2014).
- [200] D. C. C. Peixoto, R. M. Possa, R. F. Resende, and C. I. P. S. Pádua. "An overview of the main design characteristics of simulation games in Software Engineering education." In: *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEET)*. 2011, pp. 101–110. DOI: 10.1109/CSEET.2011.5876076.
- [201] D. C. C. Peixoto, R. M. Possa, R. F. Resende, and C. I. P. S. Pádua. "Challenges and issues in the development of a Software Engineering simulation game." In: *2012 Frontiers in Education Conference Proceedings*. Oct. 2012, pp. 1–6. DOI: 10.1109/FIE.2012.6462318.

- [202] D. C. C. Peixoto, R. M. Possa, R. F. Resende, and C. I. P. S. Pádua. "FASENG: A framework for development of Software Engineering simulation games." In: *2012 Frontiers in Education Conference Proceedings*. Oct. 2012, pp. 1–6. DOI: 10.1109/FIE.2012.6462319.
- [203] D. C. C. Peixoto, R. F. Resende, and C. I. P. S. Pádua. "An educational simulation model derived from academic and industrial experiences." In: *2013 IEEE Frontiers in Education Conference (FIE)*. Oct. 2013, pp. 691–697. DOI: 10.1109/FIE.2013.6684914.
- [204] D. C. C. Peixoto, R. F. Resende, and C. I. P. S. Pádua. "Evaluating software engineering simulation games: The UGALCO framework." In: *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. Oct. 2014, pp. 1–9. DOI: 10.1109/FIE.2014.7044204.
- [205] D. C. C. Peixoto, R. F. Resende, and C. I. P. S. Pádua. "The issues of adopting simulation games in software engineering classes." In: *IEEE Frontiers in Education Conference (FIE), 2015*. 32614 2015. 2015, pp. 1–8. DOI: 10.1109/FIE.2015.7344071.
- [206] Cécile Péraire and Todd Sedano. "Essence Reflection Meetings: Field Study." In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE '14. New York, NY, USA: ACM, 2014, 25:1–25:4. DOI: 10.1145/2601248.2601296.
- [207] Cécile Péraire and Todd Sedano. "State-Based Monitoring and Goal-Driven Project Steering: Field Study of the SEMAT Essence Framework." In: *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM. Hyderabad, India: ACM, 2014, pp. 325–334. DOI: 10.1145/2591062.2591155.
- [208] Cécile Péraire and Carlos Zapata. "A Step Forward in Software Engineering Education: Introducing the SEMAT Essence Framework, Keynote." In: *LACREST 2013 - LATIN AMERICAN CONGRESS ON REQUIREMENTS ENGINEERING & SOFTWARE TESTING*. Medellín, Colombia, Dec. 2013. URL: <http://docslide.us/technology/semat-se-education-lacrest-2013-keynote.html> (visited on 10/13/2015).
- [209] Dewayne E. Perry and Don Batory. "A Theory About the Structure of GTSEs." In: *Proceedings of the Fourth SEMAT Workshop on General Theory of Software Engineering*. GTSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 39–46.
- [210] Dietmar Pfahl, Nataliya Koval, and Günther Ruhe. "An experiment for evaluating the effectiveness of using a system dynamics simulation model in software project management ed-

- ucation." In: *Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International*. IEEE, 2001, pp. 97–109.
- [211] Dietmar Pfahl, Oliver Laitenberger, Jörg Dorsch, and Günther Ruhe. "An externally replicated experiment for evaluating the learning effectiveness of using simulations in software project management education." In: *Empirical software engineering* 8.4 (2003), pp. 367–395. URL: <http://link.springer.com/article/10.1023/A:1025320418915> (visited on 09/09/2016).
- [212] Dietmar Pfahl, Oliver Laitenberger, Günther Ruhe, Jörg Dorsch, and Tatyana Krivobokova. "Evaluating the learning effectiveness of using simulations in software project management education: results from a twice replicated experiment." In: *Information and software technology* 46.2 (2004), pp. 127–147. URL: <http://www.sciencedirect.com/science/article/pii/S0950584903001150> (visited on 09/09/2016).
- [213] Jöran Pieper. "Learning software engineering processes through playing games." In: *2012 2nd International Workshop on Games and Software Engineering (GAS)*. Zurich, Switzerland, Sept. 2012, pp. 1–4. DOI: 10.1109/GAS.2012.6225921. URL: <http://dx.doi.org/10.1109/GAS.2012.6225921>.
- [214] Jöran Pieper. "Discovering the essence of Software Engineering - an integrated game-based approach based on the SE-MAT Essence specification." In: *2015 IEEE Global Engineering Education Conference (EDUCON)*. Mar. 2015, pp. 939–947. DOI: 10.1109/EDUCON.2015.7096086.
- [215] Mathias Ekstedt Pontus Johnson. "Towards general theories of software engineering." In: *Science of Computer Programming* 101 (2015). DOI: 10.1016/j.scico.2014.11.005.
- [216] H. Pötter, M. Schots, L. Duboc, and V. Werneck. "InspectorX: A game for software inspection training and learning." In: *2014 IEEE 27th Conference on Software Engineering Education and Training (CSEET)*. Apr. 2014, pp. 55–64. DOI: 10.1109/CSEET.2014.6816782.
- [217] Marc Prensky. "'Engage Me or Enrage Me': What Today's Learners Demand." In: *Educause review* 40.5 (2005), pp. 60–65. URL: <http://cff.wiki.elanco.net/file/view/Engage+Me+or+Enrage+Me.pdf/84842561/Engage+Me+or+Enrage+Me.pdf> (visited on 02/04/2014).
- [218] Marc Prensky. *Digital game-based learning*. Paragon House, 2007.
- [219] Project Management Institute, Inc. *PMBOK - Guide and Standards*. 2016. URL: <https://www.pmi.org/pmbok-guide-standards> (visited on 10/31/2016).

- [220] David Raffo and Joseph Vandeville. "Combining Process Feedback with Discrete Event Simulation Models to Support Software Project Management." In: *Software evolution and feedback: Theory and Practice*. Ed. by N. H. Madhavji, J. C. Fernández-Ramil, and D. E. Perry. 1st ed. Chichester, UK: John Wiley & Sons, Ltd, 2006, pp. 427–442.
- [221] Ragna Steenweg, Marco Kuhrmann, and Daniel Méndez Fernández. *Software Engineering Process Metamodels: A Literature Review*. 2012. URL: <https://mediatum.ub.tum.de/attfile/1128389/hd2/incoming/2012-Dec/154155.pdf>.
- [222] Paul Ralph, Gregor Engels, Ivar Jacobson, and Michael Goedicke. "4th SEMAT Workshop on General Theory of Software Engineering (GTSE 2015)." In: *Proceedings of the 37th International Conference on Software Engineering - Volume 2. ICSE '15*. Piscataway, NJ, USA: IEEE Press, 2015, pp. 983–984.
- [223] Kersten Reich. *Konstruktivistische Didaktik: Das Lehr- und Studienbuch mit Online-Methodenpool*. 5., erweiterte Aufl. Beltz, June 2012.
- [224] Charles M. Reigeluth. "In search of a better way to organize instruction: The elaboration theory." en. In: *Journal of instructional development* 2.3 (1979), pp. 8–15. DOI: 10.1007/BF02984374.
- [225] Charles M. Reigeluth. "The elaboration theory: Guidance for scope and sequence decisions." In: *Instructional-design theories and models: A new paradigm of instructional theory*. Ed. by C. M. Reigeluth. Vol. 2. Mahwah, NJ: Lawrence Erlbaum Associates, 1999, pp. 425–453.
- [226] Ricardo Balduino. *Introduction to OpenUP (Open Unified Process)*. Aug. 2007. URL: <http://www.eclipse.org/epf/general/OpenUP.pdf> (visited on 11/08/2016).
- [227] Lloyd P. Rieber. "Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games." In: *Educational Technology Research and Development* 44.2 (June 1996), pp. 43–58. DOI: 10.1007/BF02300540.
- [228] Ute Ritterfeld, Michael J. Cody, and Peter Vorderer. *Serious Games: Mechanisms and Effects*. Taylor & Francis, June 2009.
- [229] Ute Ritterfeld and René Weber. "Video games for entertainment and education." In: *Playing video games: Motives, responses, and consequences*. Ed. by P. Vorderer and J. Bryant. Mahwah, NJ: Lawrence Erlbaum Associates, 2006, pp. 399–413.
- [230] Robert M. Gagne and Karen L. Medsker. *The Conditions of Learning: Training Applications*. Englisch. Fort Worth: Harcourt Brace, 1996.

- [231] Daniel Rodriguez, Miguel Angel Sicilia, Juan Jose Cuadrado-Gallego, and Dietmar Pfahl. "e-learning in project management using simulation models: A case study based on the replication of an experiment." In: *IEEE Transactions on Education* 49.4 (2006), pp. 451–463.
- [232] Romain Franceschini, Paul-Antoine Bisgambiglia, Paul-Antoine Bisgambiglia, and David Hill. "DEVS-Ruby: a Domain Specific Language for DEVS Modeling and Simulation)." In: *DEVS '14 Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*. Tampa, Florida, 2014, pp. 393–398.
- [233] Iván Ruiz-Rube, Juan Manuel Dodero, Manuel Palomo-Duarte, Mercedes Ruiz, and David Gawn. "Uses and applications of spem process models. A systematic mapping study." In: *Journal of Software Maintenance and Evolution: Research and Practice* 1.32 (2012), pp. 999–1025.
- [234] Gavriel Salomon. "Television is "easy" and print is "tough": The differential investment of mental effort in learning as a function of perceptions and attributions." In: *Journal of Educational Psychology* 76.4 (1984), pp. 647–658. DOI: 10.1037/0022-0663.76.4.647.
- [235] Hildegard Schaeper and Kolja Briedis. *Kompetenzen von Hochschulabsolventinnen und Hochschulabsolventen, berufliche Anforderungen und Folgerungen für die Hochschulreform: Projektbericht*. HIS-Hochschul-Informationen-System, 2004.
- [236] Roger C. Schank. *Virtual Learning: A Revolutionary Approach to Building a Highly Skilled Workforce*. English. 1st edition. New York: McGraw-Hill, May 1997.
- [237] Jesse Schell. *The Art of Game Design: A Book of Lenses*. Morgan Kaufmann, Sept. 2008.
- [238] H. G Schmidt, S. M.M Loyens, T. Van Gog, and F. Paas. "Problem-based learning is compatible with human cognitive architecture: Commentary on Kirschner, Sweller, and Clark (2006)." In: *Educational Psychologist* 42.2 (2007), pp. 91–97.
- [239] K. Schneider, O. Liskin, H. Paulsen, and S. Kauffeld. "Requirements compliance as a measure of project success." In: *2013 IEEE Global Engineering Education Conference (EDUCON)*. Mar. 2013, pp. 1276–1283. DOI: 10.1109/EduCon.2013.6530271.
- [240] Dale H. Schunk. *Learning Theories: An Educational Perspective*. English. 6 edition. Boston: Pearson, Jan. 2011.
- [241] Y. Sedelmaier and D. Landes. "Evaluating didactical approaches based upon students' competences." In: *2016 IEEE Global Engineering Education Conference (EDUCON)*. Apr. 2016, pp. 527–536. DOI: 10.1109/EDUCON.2016.7474603.

- [242] Yvonne Sedelmaier, Sascha Claren, and Dieter Landes. "Welche Kompetenzen benötigt ein Software Ingenieur?" In: *SEUH 2013 - Software Engineering im Unterricht der Hochschulen 2013*. Vol. 956. Aachen: CEUR Workshop Proceedings (CEUR-WS.org), 2013, pp. 117–128. DOI: urn:nbn:de:0074-956-4.
- [243] Yvonne Sedelmaier and Dieter Landes. "Software engineering body of skills (SWEBOS)." In: *Global Engineering Education Conference (EDUCON), 2014 IEEE*. Istanbul, Turkey: IEEE, Apr. 2014, pp. 395–401. DOI: 10.1109/EDUCON.2014.6826125.
- [244] SEMAT. *Kickstarting a Project - an educational case study*. URL: <http://semat.org/documents/20181/27952/Kickstarting-a-Project.pdf> (visited on 07/20/2016).
- [245] SEMAT. *Software Engineering Method and Theory*. 2014. URL: <http://semat.org/> (visited on 10/21/2014).
- [246] SEMAT, semat.org. *Essence in Practice – a Revolution in Software Engineering? SEMAT*. URL: <http://semat.org/essence-in-practice> (visited on 08/13/2016).
- [247] semat.org. *Tool Support - SEMAT*. (Archived by WebCite® at <http://www.webcitation.org/6jhxxLUVk>). 2014. URL: <http://semat.org/tool-support3> (visited on 06/19/2014).
- [248] semat.org. *Frequently Asked Questions - SEMAT*. FAQ. (Archived by WebCite® at <http://www.webcitation.org/6eteXMJO1>). Jan. 2016. URL: <http://semat.org/de/frequently-asked-questions> (visited on 01/29/2016).
- [249] Helen Sharp and Pat Hall. "An interactive multimedia software house simulation for postgraduate software engineers." In: *Proceedings of the International Conference on Software Engineering (ICSE 2000)*. Limerick, Ireland: IEEE Computer Society, 2000, p. 688. DOI: 10.1109/ICSE.2000.10053.
- [250] Katherine Shaw and Julian Dermoudy. "Engendering an empathy for software engineering." In: *Proceedings of the 7th Australasian conference on Computing education - Volume 42. ACE '05*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2005, pp. 135–144.
- [251] Mary Shaw. "Software Engineering Education: A Roadmap." In: *Proceedings of the Conference on The Future of Software Engineering. ICSE '00*. New York, NY, USA: ACM, 2000, pp. 371–380. DOI: 10.1145/336512.336592.
- [252] Cuihua Shen, Hua Wang, and Ute Ritterfeld. "Serious games and seriously fun games." In: *Serious games: Mechanisms and effects*. Ed. by Ute Ritterfeld, Peter Vorderer, and Michael Cody. Routledge, 2009, pp. 49–61.

- [253] Gerry Shih. "Facebook admits year-long data breach exposed 6 million users." In: *Reuters* (June 2013). URL: <http://www.reuters.com/article/2013/06/21/net-us-facebook-security-idUSBRE95K18Y20130621> (visited on 02/21/2014).
- [254] Valerie J. Shute and Joseph Psotka. *Intelligent Tutoring Systems: Past, Present, and Future*. Tech. rep. Defense Technical Information Center, Armstrong Laboratories, Brooks Air Force Base, Texas, 1994. URL: <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA280011> (visited on 08/22/2016).
- [255] Valerie J. Shute, Matthew Ventura, Malcolm Bauer, and Diego Zapata-Rivera. "Melding the power of serious games and embedded assessment to monitor and foster learning." In: *Serious games: Mechanisms and effects*. Ed. by Ute Ritterfeld, Peter Vorderer, and Michael Cody. Routledge, 2009, pp. 295–321.
- [256] George Siemens. "Connectivism: A learning theory for the digital age." In: *International Journal of Instructional Technology and Distance Learning (ITDL)* (2005). URL: <http://er.dut.ac.za/handle/123456789/69> (visited on 09/28/2016).
- [257] SimSE, University of California, Irvine. *SimSE Downloads*. 2010. URL: <http://www.ics.uci.edu/~emilyo/SimSE/downloads.html> (visited on 09/09/2016).
- [258] Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, U.S.A. *CMMI for Development, Version 1.3*. Nov. 2010. URL: <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9661> (visited on 10/31/2016).
- [259] Elliot Soloway, Shari L. Jackson, Jonathan Klein, Chris Quintana, James Reed, Jeff Spitulnik, Steven J. Stratford, Scott Studer, Jim Eng, and Nancy Scala. "Learning Theory in Practice: Case Studies of Learner-centered Design." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '96. New York, NY, USA: ACM, 1996, pp. 189–196. DOI: 10.1145/238386.238476.
- [260] Ian Sommerville. *Software Engineering*. 9th revised edition. Addison-Wesley Longman, Amsterdam, Feb. 2010.
- [261] A. Soska, J. Mottok, and C. Wolff. "An experimental card game for software testing: Development, design and evaluation of a physical card game to deepen the knowledge of students in academic software testing education." In: *2016 IEEE Global Engineering Education Conference (EDUCON)*. Abu Dhabi, United Arab Emirates, Apr. 2016, pp. 576–584. DOI: 10.1109/EDUCON.2016.7474609.

- [262] Alexander Soska, Jürgen Mottok, and Christian Wolff. "Playful learning in academic software engineering education." In: *2015 IEEE Global Engineering Education Conference (EDUCON)*. Tallinn, Estonia: IEEE, 2015, pp. 324–332. DOI: 10.1109/EDUCON.2015.7095992.
- [263] K. Squire. "From Content to Context: Videogames as Designed Experience." en. In: *Educational Researcher* 35.8 (Nov. 2006), pp. 19–29. DOI: 10.3102/0013189X035008019.
- [264] Alice Squires, Wiley Larson, and Brian Sauser. "Mapping space-based systems engineering curriculum to government-industry vetted competencies for improved organizational performance." en. In: *Systems Engineering* 13.3 (2009), pp. 246–260. DOI: 10.1002/sys.20146.
- [265] Herbert Stachowiak. *Allgemeine Modelltheorie*. Wien, New York: Springer, 1973.
- [266] Statista. *WhatsApp: age distribution of users in the United States 2014* | Statistic. URL: <http://www.statista.com/statistics/290447/age-distribution-of-us-whatsapp-users/> (visited on 08/16/2016).
- [267] John D. Sterman. *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Har/Cdr. Mcgraw-Hill Higher Education, 2000.
- [268] John D. Sterman. "All models are wrong: reflections on becoming a systems scientist." In: *System Dynamics Review* 18.4 (2002), pp. 501–531. DOI: 10.1002/sdr.261.
- [269] Stevens Institute of Technology. *Graduate Software Engineering 2009 (GSWE2009): Curriculum Guidelines for Graduate Degree Programs in Software Engineering*. Sept. 2009.
- [270] Angela Stoof, Rob L. Martens, Jeroen J. G. van Merriënboer, and Theo J. Bastiaens. "The Boundary Approach of Competence: A Constructivist Aid for Understanding and Using the Concept of Competence." en. In: *Human Resource Development Review* 1.3 (Sept. 2002), pp. 345–365. DOI: 10.1177/1534484302013005.
- [271] R. Studt, G. Winterfeldt, and J. Mottok. "Measuring software engineering competencies." In: *2015 IEEE Global Engineering Education Conference (EDUCON)*. Mar. 2015, pp. 908–914. DOI: 10.1109/EDUCON.2015.7096081.
- [272] Patrick Suppes. "The place of theory in educational research." In: *Educational Researcher* 3.6 (1974), pp. 3–10.
- [273] John Sweller. "Cognitive load during problem solving: Effects on learning." In: *Cognitive Science* 12.2 (June 1988), pp. 257–285. DOI: 10.1207/s15516709cog1202_4.

- [274] John Sweller. "Some cognitive processes and their consequences for the organisation and presentation of information." In: *Australian Journal of Psychology* 45.1 (1993), pp. 1–8. DOI: 10.1080/00049539308259112.
- [275] Maureen Tam. "Constructivism, instructional design, and technology: Implications for transforming distance learning." In: *Educational Technology & Society* 3.2 (2000), pp. 50–60.
- [276] Gil Taran. "Using games in software engineering education to teach risk management." In: *20th Conference on Software Engineering Education & Training (CSEET'07)*. IEEE, 2007, pp. 211–220. DOI: 10.1109/CSEET.2007.54.
- [277] The OECD Program Definition and Selection of Competencies: Theoretical and Conceptual Foundations (DeSeCo). *The Definition And Selection of Key Competencies : Executive Summary*. July 2005. URL: <http://www.oecd.org/edu/skills-beyond-school/definitionandselectionofcompetenciesdeseco.htm> (visited on 08/26/2016).
- [278] The Ruby Community. *Ruby Programming Language*. URL: <https://www.ruby-lang.org/en/> (visited on 08/13/2016).
- [279] The Ruby On Rails Community. *Ruby on Rails*. URL: <http://rubyonrails.org/> (visited on 08/13/2016).
- [280] The SEMAT Community. *SEMAT - Alpha State Cards*. 2016. URL: <http://semat.org/de/alpha-state-cards-with-abbrev-checklists> (visited on 08/15/2016).
- [281] The Standish Group. *THE CHAOS REPORT*. 1994.
- [282] The Standish Group. *CHAOS MANIFESTO 2013 - Think Big, Act Small*. 2013.
- [283] Thomas Hilburn, Mark Ardis, Glenn Johnson, Andrew Kornecki, and Nancy R. Mead. *Software Assurance Competency Model*. Mar. 2013. URL: <http://www.sei.cmu.edu/reports/13tn004.pdf> (visited on 08/26/2016).
- [284] Veronika Thurner, Axel Böttcher, Kathrin Schlierkamp, and Daniela Zehetmeier. "Lernziele für die Kompetenzentwicklung auf höheren Taxonomiestufen." In: *Software Engineering im Unterricht der Hochschulen (SEUH)* (2015), pp. 9–20.
- [285] Time Magazine / CBS News. "Papert on Piaget." en. In: *People of the Century*. New York: Simon and Schuster, 1999, p. 105.
- [286] C. Timothy. "What knowledge is important to a software professional?" In: *Computer* 33.5 (May 2000), pp. 44–50. DOI: 10.1109/2.841783.
- [287] Bruce W. Tuckman and Mary Ann C. Jensen. "Stages of small-group development revisited." In: *Group & Organization Management* 2.4 (1977), pp. 419–427. DOI: 10.1177/105960117700200404.

- [288] Richard T. Turley and James M. Bieman. "Identifying Essential Competencies of Software Engineers." In: *ACM Conference on Computer Science*. 1994, pp. 271–278.
- [289] Richard T. Turley and James M. Bieman. "Competencies of exceptional and nonexceptional software engineers." In: *Journal of Systems and Software* 28.1 (1995), pp. 19–38. DOI: 10.1016/0164-1212(94)00078-2.
- [290] Universität Stuttgart, Informatik, Institute of Software Technology (ISTE), Research, SESAM. *SESAM - Modeling Language*. Nov. 2010. URL: <http://www.iste.uni-stuttgart.de/en/seuntil-march-2013/research/areas-of-research-and-projects/sesam-software-engineering-simulation-by-animated-models/sesam-modeling-language.html> (visited on 09/10/2016).
- [291] M. Utesch, R. Heininger, and H. Krcmar. "Strengthening study skills by using ERPsims as a new tool within the Pupils' academy of serious gaming." In: *2016 IEEE Global Engineering Education Conference (EDUCON)*. Apr. 2016, pp. 592–601. DOI: 10.1109/EDUCON.2016.7474611.
- [292] M. Utesch, R. Heininger, and H. Krcmar. "The pupils' academy of serious gaming: Strengthening study skills with ERPsims." In: *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*. Feb. 2016, pp. 93–102. DOI: 10.1109/REV.2016.7444446.
- [293] D. Valencia, A. Vizcaíno, L. Garcia-Mundo, M. Piattini, and J. P. Soto. "GSDgame: A Serious Game for the Acquisition of the Competencies Needed in GSD." In: *2016 IEEE 11th International Conference on Global Software Engineering Workshops (ICGSEW)*. Aug. 2016, pp. 19–24. DOI: 10.1109/ICGSEW.2016.11.
- [294] Ton Van Der Valk, Jan H. Van Driel, and Wobbe De Vos. "Common Characteristics of Models in Present-day Scientific Practice." en. In: *Research in Science Education* 37.4 (Jan. 2007), pp. 469–488. DOI: 10.1007/s11165-006-9036-3.
- [295] Richard Van Eck. "Digital game-based learning: It's not just the digital natives who are restless." In: *Educause review* 41.2 (2006), pp. 16–30.
- [296] Katia Vega, Hugo Fuks, and Gustavo Carvalho. "Training in Requirements by Collaboration: Branching Stories in Second Life." In: *2009 Simposio Brasileiro de Sistemas Colaborativos*. IEEE, 2009, pp. 116–122. DOI: 10.1109/SBSC.2009.11.
- [297] VersionOne, Inc. *The 10th Annual State of Agile™ Report | VersionOne*. URL: <http://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf> (visited on 08/15/2016).

- [298] Christiane Gresse Von Wangenheim, Marcello Thiry, and Djone Kochanski. "Empirical evaluation of an educational game on software measurement." In: *Empirical Software Engineering* 14.4 (2009), pp. 418–452. DOI: 10.1007/s10664-008-9092-6.
- [299] Alf Inge Wang, Terje Øfsdahl, and Ole Kristian Mørch-Storstein. "An evaluation of a mobile game concept for lectures." In: *Software Engineering Education and Training, 2008. CSEET'08. IEEE 21st Conference on*. IEEE, 2008, pp. 197–204. DOI: 10.1109/CSEET.2008.15.
- [300] Xiaofeng Wang, Kieran Conboy, and Oisin Cawley. "'Leagile' software development: An experience report analysis of the application of lean approaches in agile software development." In: *Journal of Systems and Software*. Special Issue: Agile Development 85.6 (June 2012), pp. 1287–1299. DOI: 10.1016/j.jss.2012.01.061.
- [301] Christiane Gresse von Wangenheim and Forrest Shull. "To Game or Not to Game?" In: *IEEE Software* 26.2 (2009), pp. 92–94.
- [302] *WhatsApp - Anteil der Nutzer nach Altersgruppen in Deutschland 2015 | Umfrage*. URL: <http://de.statista.com/statistik/daten/studie/510985/umfrage/anteil-der-nutzer-von-whatsapp-nach-altersgruppen-in-deutschland/> (visited on 08/17/2016).
- [303] *WhatsApp: number of users 2013-2016 | Statistic*. URL: <http://www.statista.com/statistics/260819/number-of-monthly-active-whatsapp-users/> (visited on 08/16/2016).
- [304] Wikipedia Community. *Transporter (Star Trek)*. en. Page Version ID: 732694929. Aug. 2016. URL: [https://en.wikipedia.org/w/index.php?title=Transporter_\(Star_Trek\)&oldid=732694929](https://en.wikipedia.org/w/index.php?title=Transporter_(Star_Trek)&oldid=732694929) (visited on 09/25/2016).
- [305] William Winn and Daniel Snyder. "Cognitive perspectives in psychology." In: *Handbook of research for educational communications and technology: A project of the Association for Educational Communications and Technology* (1996), pp. 79–112.
- [306] En Ye, Chang Liu, and Jennifer A. Polack-Wahl. "Enhancing software engineering education using teaching aids in 3-D online virtual worlds." In: *2007 37th Annual Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports*. Milwaukee, WI, USA: IEEE, 2007, T1E-8-T1E-13. DOI: 10.1109/FIE.2007.4417884.
- [307] A. Zeid. "Using simulation games to teach global software engineering courses." In: *IEEE Frontiers in Education Conference (FIE), 2015. 32614 2015*. Oct. 2015, pp. 1–9. DOI: 10.1109/FIE.2015.7344110.

- [308] B. P. Zeigler. *Theory of Modelling and Simulation*. New York: Editions John Wiley, 1976.
- [309] Bernard P. Zeigler. "Hierarchical, modular discrete-event modelling in an object-oriented environment." In: *SIMULATION* 49.5 (1987), pp. 219–230. DOI: 10.1177/003754978704900506.
- [310] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of Modeling and Simulation, Second Edition*. 2nd ed. Elsevier Academic Press, Jan. 2000.
- [311] He Zhang, Ross Jeffery, Dan Houston, Liguang Huang, and Liming Zhu. "Impact of Process Simulation on Software Practice: An Initial Report." In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. New York, NY, USA: ACM, 2011, pp. 1046–1056. DOI: 10.1145/1985793.1985993.
- [312] He Zhang, Barbara Kitchenham, and Dietmar Pfahl. "Reflections on 10 Years of Software Process Simulation Modeling: A Systematic Review." In: *Proceedings of the Software Process, 2008 International Conference on Making Globally Distributed Software Development a Success Story*. ICSP'o8. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 345–356.
- [313] He Zhang, David Raffo, Thomas Birkhölzer, Dan Houston, Raymond Madachy, Jürgen Münch, and Stanley M Sutton. "Software process simulation—at a crossroads?" en. In: *Journal of Software: Evolution and Process* 26.10 (Oct. 2014), pp. 923–928. DOI: 10.1002/smr.1694.
- [314] Gabe Zichermann and Christopher Cunningham. *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*. 1st ed. O'Reilly Media, Aug. 2011.

CODING SCHEME OF ESSENCE KERNEL ELEMENTS USED IN GRAPHS

To enable a compact representation of Essence kernel's elements in the form of graphs, a simple coding scheme was used to make the elements of interest identifiable.

A.1 ALPHA IDENTIFIERS

Table A.1 shows coding of Essence kernel's 7 Alphas. Alphas are coded as a single-figure number.

Table A.1: Alpha Identifier Coding Scheme Used in Graphs

ALPHA	AREA OF CONCERN	IDENTIFIER
Stakeholders	Customer	1
Opportunity	Customer	2
Requirements	Solution	3
Software System	Solution	4
Team	Endeavor	5
Work	Endeavor	6
Way of Working	Endeavor	7

A.2 ALPHA STATE IDENTIFIERS

Table A.2 shows coding of Essence kernel's 41 Alpha States. Alpha States are coded as a two-figure number with a first figure representing the Alpha and a second figure representing the successive Alpha State of that Alpha.

Table A.2: Alpha State Identifier Coding Scheme Used in Graphs

ALPHA	ALPHA STATE	IDENTIFIER
Stakeholders	Recognized	11
Stakeholders	Represented	12

continued on the next page...

Table A.2: Alpha State Identifier Coding Scheme Used in Graphs

ALPHA	ALPHA STATE	IDENTIFIER
Stakeholders	Involved	13
Stakeholders	In Agreement	14
Stakeholders	Satisfied for Deployment	15
Stakeholders	Satisfied in Use	16
Opportunity	Identified	21
Opportunity	Solution Needed	22
Opportunity	Value Established	23
Opportunity	Viable	24
Opportunity	Addressed	25
Opportunity	Benefit Accrued	26
Requirements	Conceived	31
Requirements	Bounded	32
Requirements	Coherent	33
Requirements	Acceptable	34
Requirements	Addressed	35
Requirements	Fulfilled	36
Software System	Architecture Selected	41
Software System	Demonstrable	42
Software System	Usable	43
Software System	Ready	44
Software System	Operational	45
Software System	Retired	46
Team	Seeded	51
Team	Formed	52
Team	Collaborating	53
Team	Performing	54

continued on the next page...

Table A.2: Alpha State Identifier Coding Scheme Used in Graphs

ALPHA	ALPHA STATE	IDENTIFIER
Team	Adjourned	55
Work	Initiated	61
Work	Prepared	62
Work	Started	63
Work	Under Control	64
Work	Concluded	65
Work	Closed	66
Way of Working	Principles Established	71
Way of Working	Foundation Established	72
Way of Working	In Use	73
Way of Working	In Place	74
Way of Working	Working Well	75
Way of Working	Retired	76

A.3 CHECKPOINT IDENTIFIERS

Checkpoints were coded as a four-figure number, with the first two figures representing the corresponding Alpha State (A.2) and the last two figures representing the Checkpoint (with a trailing null if the number of the checkpoint was single-figured). Because of the high number of Checkpoints a table is not presented here. Checkpoints and resulting relationships of their Alpha States get discussed in B. There each Checkpoint of the Essence Kernel gets presented with its corresponding code used in graphs presented throughout the thesis.

A.4 ACTIVITY SPACE IDENTIFIERS

Table A.3 shows coding of Essence kernel's 15 ActivitySpaces. Activity Spaces are coded as a four-figure number, starting with 99 to distinguish them clearly from Alpha States and Checkpoints and ending with two figures representing the respective Activity Space. This coding scheme serves solely for referencing purposes, to enable a compact representation in graphs shown throughout the thesis. It should not create the impression that the Activity Spaces would have to be sequenced in a way corresponding to that numbering. A reasonable

sequencing is defined by the progressing Alpha States that are addressed by the respective Activity Space. Where appropriate Activity Spaces are colored by the color codes used in the Essence specification, where green stands for *customer*, yellow for *solution*, and blue for *endeavor*. [188]

Table A.3: Activity Space Identifier Coding Scheme Used in Graphs

ACTIVITY SPACE	AREA OF CONCERN	IDENTIFIER
Explore Possibilities	Customer	9901
Understand Stakeholders Needs	Customer	9902
Ensure Stakeholders Satisfaction	Customer	9903
Use the System	Customer	9904
Understand the Requirements	Solution	9905
Shape the System	Solution	9906
Implement the System	Solution	9907
Test the System	Solution	9908
Deploy the System	Solution	9909
Operate the System	Solution	9910
Prepare to do the Work	Endeavor	9911
Coordinate Activity	Endeavor	9912
Support the Team	Endeavor	9913
Track Progress	Endeavor	9914
Stop the Work	Endeavor	9915

ANALYSIS OF ALPHAS' INTERDEPENDENCIES IN THE SEMAT ESSENCE KERNEL—MAKING IMPLICIT DEPENDENCIES EXPLICIT

It has to be emphasized that it is not the goal of this work to define the ONE correct path through an SE endeavor but to define a plausible one that is easy to explain. There is not just one possible path. As already stated in 3.3 each topological sort order of the dependency graph constructed as described in 3.3 is a valid one from the perspective of the chosen modeling approach.

Given the flexibility of the SEMAT Essence Kernel its possible to interpret checkpoints differently in a context-sensitive way potentially leading to another set of (inter-)dependencies. Substituting the proposed set of (inter-)dependencies below with different one and following the procedure described in 3.3 would produce a different graph and hence lead to a different set of topological sort orders defining different possible paths through an SE endeavor.

Some of the alpha states combine checkpoints with multiple influences from other Alpha States. While it is often sufficient to define one Alpha State as a precondition to another Alpha State, situations occur where one alpha state <A> depends on another alpha state which in turn depends directly or indirectly on <A>. To enable the simulation of the Essence kernel, such circular dependencies have to be avoided. This may require defining dependencies not on Alpha State level but Checkpoint level. Defining the inter-dependencies on Checkpoints level categorically was not chosen as approach since the resulting model would be much more complex without any additional benefits.

Discussing single AlphaStates and their interdependencies requires a referencing of single Checkpoints describing them. To enable the referencing of single Checkpoints of Alpha States, they are numbered in the following section. This numbering does not exist in the Essence specification[188] and does not define any sequence on them. It is used solely for identification and referencing purposes. The number in square brackets behind each of the Checkpoint's description represents the coding used to represent the respective Checkpoint in graphs used throughout the thesis (cf. A on page 325 for details).

The method to define interdependencies, or rather to make them explicit, got already described in section 3.3 on page 127. This chapter documents decisions made in the process to provide a transparent, traceable, and replicable result.

B.O.1 *Alpha: Stakeholders*B.O.1.1 *Alpha State Recognized: Stakeholders have been identified**Checkpoints:*

1. All the different groups of stakeholders that are, or will be, affected by the development and operation of the software system are identified. [1101]
2. There is agreement on the stakeholder groups to be represented. At a minimum, the stakeholders groups that fund, use, support, and maintain the system have been considered. [1102]
3. The responsibilities of the stakeholder representatives have been defined. [1103]

Targeted by Activity Space(s):

- Explore Possibilities()

Analysis:

- Checkpoints 1101 and 1102 mention a “software system” and “system” but do not (yet) put any requirements on it.

Dependencies:

- none

B.O.1.2 *Alpha State Represented: The mechanisms for involving the stakeholders are agreed and the stakeholder representatives have been appointed.**Checkpoints:*

1. The stakeholder representatives have agreed to take on their responsibilities. [1201]
2. The stakeholder representatives are authorized to carry out their responsibilities. [1202]
3. The collaboration approach among the stakeholder representatives has been agreed. [1203]
4. The stakeholder representatives support and respect the team’s way of working. [1204]

Targeted by Activity Space(s):

- Understand Stakeholder Needs(Stakeholders, Opportunity, Requirements, Software System)

Analysis:

- Checkpoint 1204 mentions Alphas *Team* and *Way of Working*. In order to respect team's way of working it has to be defined before.

Dependencies:

- CP-1204 depends on *WayOfWorking::FoundationEstablished* (requires existing *Team*, dependency gets defined at Alpha *Way of Working*)

B.O.1.3 *Alpha State Involved: The stakeholder representatives are actively involved in the work and fulfilling their responsibilities.*

Checkpoints:

1. The stakeholder representatives assist the team in accordance with their responsibilities. [1301]
2. The stakeholder representatives provide feedback and take part in decision making in a timely manner. [1302]
3. The stakeholder representatives promptly communicate changes that are relevant for their stakeholder groups. [1303]

Targeted by Activity Space(s):

- Understand Stakeholder Needs(Stakeholders, Opportunity, Requirements, Software System)

Analysis:

- Checkpoint 1301 states "assist the team". This requires an already existing team.
- Checkpoint 1302 states "provide feedback and take part in decision making". One perspective could be that Work would at least be Initiated to provide feedback etc. The perspective taken here is that feedback is already needed before (development) work is started, e.g. for requirements elicitation, etc.

Dependencies:

- depends on: *Team::Seeded*

B.O.1.4 *Alpha State In Agreement: The stakeholder representatives are in agreement.*

Checkpoints:

1. The stakeholder representatives have agreed upon their minimal expectations for the next deployment of the new system. [1401]

2. The stakeholder representatives are happy with their involvement in the work. [1402]
3. The stakeholder representatives agree that their input is valued by the team and treated with respect. [1403]
4. The team members agree that their input is valued by the stakeholder representatives and treated with respect. [1404]
5. The stakeholder representatives agree with how their different priorities and perspectives are being balanced to provide a clear direction for the team. [1405]

Targeted by Activity Space(s):

- Understand Stakeholder Needs(Stakeholders, Opportunity, Requirements, Software System)

Analysis:

- Checkpoint 1401 mentions “the new system” but puts not yet any precondition on it.
- Checkpoint 1402 mentions “... involvement in the work.” This needs *Work* to be *Started*.
- Checkpoints 1403, 1404, and 1405 mention the “team” and “team members”. This requires an existing *Team*.

Dependencies:

- depends on: *Work::Started* and
- depends on *Team::Seeded* (but this dependency was already defined by *Stakeholders::Involved*)

B.O.1.5 *Alpha State Satisfied for Deployment: The minimal expectations of the stakeholder representatives have been achieved.*

Checkpoints:

1. The stakeholder representatives provide feedback on the system from their stakeholder group perspective. [1501]
2. The stakeholder representatives confirm that they agree that the system is ready for deployment. [1502]

Targeted by Activity Space(s):

- Ensure Stakeholder Satisfaction(Stakeholders, Opportunity, Requirements, Software System)

Analysis:

- CP-1501 states “[...] provide feedback on the system [...]” This and the Alpha State (“*Satisfied for Deployment*”) itself imply that *Requirements* should already be *Addressed*.
- CP-1502 states “stakeholder representatives [...] agree that the system is ready for deployment”. This requires the *Software System* to be *Ready*.

Dependencies:

- depends on Software System::Ready
- to avoid a circular dependencies further interdependencies have to be defined on Checkpoint level
 - CP-1501 depends on
 - * CP-3501 (Requirements::Addressed),
 - * CP-3502 (Requirements::Addressed), and
 - * CP-3503 (Requirements::Addressed)
 - CP-1502 is precondition to CP-3504 (Requirements::Addressed)

B.O.1.6 *Alpha State Satisfied in Use: The system has met or exceeds the minimal stakeholder expectations.*

Checkpoints:

1. Stakeholders are using the new system and providing feedback on their experiences. [1601]
2. The stakeholders confirm that the new system meets their expectations. [1602]

Targeted by Activity Space(s):

- Use the System(Stakeholders, Opportunity, Requirements, Software System)

Analysis:

- CP-1601 states “... using the new system...” This requires the *Software System* to be *Operational*.
- CP-1601 states “... confirm that the new system meets their expectations.” This requires the *Requirements* to be *Fulfilled*.

Dependencies:

- depends on: Software System::Operational and
- depends on: Requirements::Fulfilled

B.O.2 *Alpha: Opportunity*

B.O.2.1 *Alpha State Identified: A commercial, social, or business opportunity has been identified that could be addressed by a software-based solution.*

Checkpoints:

1. An idea for a way of improving current ways of working, increasing market share, or applying a new or innovative software system has been identified. [2101]
2. At least one of the stakeholders wishes to make an investment in better understanding the opportunity and the value associated with addressing it. [2102]
3. The other stakeholders who share the opportunity have been identified. [2103]

Targeted by Activity Space(s):

- Explore Possibilities()

Analysis:

- CP-2102 mentions “current ways of working” and “innovative software system.” The first one does not represent the *Alpha Way of Working* and the latter one does not yet put any requirements on the *Alpha Software System*.
- CP-2102 and CP2103 mention “one of the stakeholders” and “other stakeholders.” This requires *Stakeholders* to be *Recognized*.

Dependencies:

- depends on: Stakeholders::Recognized

B.O.2.2 *Alpha State Solution Needed: The need for a software-based solution has been confirmed.*

Checkpoints:

1. The stakeholders in the opportunity and the proposed solution have been identified. [2201]
2. The stakeholders’ needs that generate the opportunity have been established. [2202]
3. Any underlying problems and their root causes have been identified. [2203]
4. It has been confirmed that a software-based solution is needed. [2204]

5. At least one software-based solution has been proposed. [2205]

Targeted by Activity Space(s):

- Explore Possibilities()

Analysis:

- CP-2201 mentions “stakeholders ... identified.” This requires the *Stakeholders* to be *Recognized*.
- CP-2204 and CP-2205 mention a “software based solution,” which may represent the *Software System Alpha*. The SEMAT Essence FAQ state “A solution can be proposed on the basis of a very sketchy understanding, while the outlined solution suggests that work has been performed to go beyond the initial proposal.”[248] Following this statement, it is assumed that the solution is proposed on a “very sketchy understanding”. This puts not yet any requirements on the *Software System Alpha*.

Dependencies:

- depends on: Stakeholders::Recognized (but this was already defined in Opportunity::Identified)

B.0.2.3 *Alpha State Value Established: The value of a successful solution has been established.*

Checkpoints:

1. The value of addressing the opportunity has been quantified either in absolute terms or in returns or savings per time period (e.g., per annum). [2301]
2. The impact of the solution on the stakeholders is understood. [2302]
3. The value that the software system offers to the stakeholders that fund and use the software system is understood. [2303]
4. The success criteria by which the deployment of the software system is to be judged are clear. [2304]
5. The desired outcomes required of the solution are clear and quantified. [2305]

Targeted by Activity Space(s):

- Explore Possibilities()

Analysis:

- CP-2302 and CP-2303 mention “stakeholders”. This requires the *Stakeholders* to be *Recognized*.

Dependencies:

- depends on: Stakeholders::Recognized (but this was already defined in Opportunity::Identified)

B.O.2.4 *Alpha State Viable: It is agreed that a solution can be produced quickly and cheaply enough to successfully address the opportunity.*

Checkpoints:

1. A solution has been outlined. [2401]
2. The indications are that the solution can be developed and deployed within constraints. [2402]
3. The risks associated with the solution are acceptable and manageable. [2403]
4. The indicative (ball-park) costs of the solution are less than the anticipated value of the opportunity. [2404]
5. The reasons for the development of a software-based solution are understood by all members of the team. [2405]
6. It is clear that the pursuit of the opportunity is viable. [2406]

Targeted by Activity Space(s):

- Understand Stakeholder Needs(Stakeholders, Opportunity, Requirements, Software System)

Analysis:

- All Checkpoints imply that the architecture of the *Software System* is already selected.
- All Checkpoints but CP-2405 imply that *Requirements* are *Bounded* and *Coherent*. Otherwise, it would hardly be possible to assess risks and estimate costs of the endeavor.
- All Checkpoints but CP-2405 imply that *Work* is already *Prepared* since this ensures that cost and effort are already estimated, and resource availability, as well as risks, are already understood.
- CP-2405 mentions “all members of the team.” This requires the *Team* to be already *Formed*.
- The SEMAT Essence FAQ states “The last checklist item could well be redundant – no additional evidence is required to mark it up as having been satisfied.”[248]

Dependencies:

- depends on SoftwareSystem::ArchitectureSelected,
- depends on Work::Prepared (maybe only partially),
- depends on Team::Formed, and
- depends on Requirements::Bounded/Coherent

B.O.2.5 *Alpha State Addressed: A solution has been produced that demonstrably addresses the opportunity.*

Checkpoints:

1. A usable system that demonstrably addresses the opportunity is available. [2501]
2. The stakeholders agree that the available solution is worth deploying. [2502]
3. The stakeholders are satisfied that the solution produced addresses the opportunity. [2503]

Targeted by Activity Space(s):

- Ensure Stakeholder Satisfaction(Stakeholders, Opportunity, Requirements, Software System)

Analysis:

- CP-2501 implies that *Software System* is *Ready*.
- CP-2502 implies that *Stakeholders* are *Satisfied For Deployment*.
- CP-2503 implies that *Requirements* are *Addressed*.

Dependencies:

- depends on SoftwareSystem::Ready,
- depends on Stakeholders::SatisfiedForDeployment, and
- depends on Requirements::Addressed

B.O.2.6 *Alpha State Benefit Accrued: The operational use or sale of the solution is creating tangible benefits.*

Checkpoints:

1. The solution has started to accrue benefits for the stakeholders. [2601]
2. The return-on-investment profile is at least as good as anticipated. [2602]

Targeted by Activity Space(s):

- Use the System(Stakeholders, Opportunity, Requirements, Software System)

Analysis:

- In order to accrue benefits at least as good as anticipated the *Software System* has to be *Operational*.

Dependencies:

- depends on: SoftwareSystem::Operational

B.O.3 *Alpha: Requirements*

B.O.3.1 *Alpha State Conceived: The need for a new system has been agreed.*

Checkpoints:

1. The initial set of stakeholders agrees that a system is to be produced. [3101]
2. The stakeholders that will use the new system are identified. [3102]
3. The stakeholders that will fund the initial work on the new system are identified. [3103]
4. There is a clear opportunity for the new system to address. [3104]

Targeted by Activity Space(s):

- Understand the Requirements(Stakeholders, Opportunity, Requirements, Software System, Work, Way-of-Working)

Analysis:

- CP-3101 mentions “initial set of stakeholders,” CP-3102 and CP-3103 mention “stakeholders ... identified”. This implies *Stakeholders* already have to be *Recognized*.
- CP-3104 mentions “clear opportunity,” This implies an *Identified Opportunity*.
- CP-3101 mentions a “system... to be produced,” CP-3104 mentions “the new system.” Both reference the *Software System Alpha* but do not yet put any conditions on it.

Dependencies:

- depends on: Stakeholders::Recognized and
- depends on: Opportunity::Identified

B.O.3.2 *Alpha State Bounded: The purpose and extent of the new system are clear.*

Checkpoints:

1. The stakeholders involved in developing the new system are identified. [3201]
2. The stakeholders agree on the purpose of the new system. [3202]
3. It is clear what success is for the new system. [3203]

4. The stakeholders have a shared understanding of the extent of the proposed solution. [3204]
5. The way the requirements will be described is agreed upon. [3205]
6. The mechanisms for managing the requirements are in place. [3206]
7. The prioritization scheme is clear. [3207]
8. Constraints are identified and considered. [3208]
9. Assumptions are clearly stated. [3209]

Targeted by Activity Space(s):

- Understand the Requirements(Stakeholders, Opportunity, Requirements, Software System, Work, Way-of-Working)

Analysis:

- CP-3201, CP-3202, and CP-3203 mention “the new system” but do not yet set any conditions on *Alpha Software System*.
- CP-3201: states “[...] stakeholders involved [...] are identified.” This implies that *Stakeholders* are already *Recognized*.
- CP-3202 and CP-3203 mention “[...] purpose of the new system.” and “[...] shared understanding of the extent of the proposed solution.” This implies that “[...] impact of the solution [...] is understood, [...] value that the software system offers [...] is understood, [...] success criteria [...] are clear,” and “[...] desired outcomes [...] are clear and quantified,” which are all descriptions of Checkpoints of *Opportunity::ValueEstablished*. This implies that the value of the *Opportunity* is already *established*.
- CP-3205, CP-3206, and CP-3207 imply that the way, requirements are described, managed, and prioritized, is already established. Hence the *foundation* of the *Way of Working*, at least with regards to *Requirements* elicitation, is *established*.

Dependencies:

- depends on *Opportunity::ValueEstablished*,
- depends on *Stakeholders::Recognized* (but dependency is already defined in predecessor alpha state *Conceived*), and
- depends on *WayOfWorking::FoundationEstablished*

B.O.3.3 *Alpha State Coherent: The requirements provide a consistent description of the essential characteristics of the new system.*

Checkpoints:

1. The requirements are captured and shared with the team and the stakeholders. [3301]
2. The origin of the requirements is clear. [3302]
3. The rationale behind the requirements is clear. [3303]
4. Conflicting requirements are identified and attended to. [3304]
5. The requirements communicate the essential characteristics of the system to be delivered. [3305]
6. The most important usage scenarios for the system can be explained. [3306]
7. The priority of the requirements is clear. [3307]
8. The impact of implementing the requirements is understood. [3308]
9. The team understands what has to be delivered and agrees to deliver it. [3309]

Targeted by Activity Space(s):

- Understand the Requirements(Stakeholders, Opportunity, Requirements, Software System, Work, Way-of-Working)

Analysis:

- CP-3301 and CP-3309 state “[...] shared with the team [...]” and “[...] team understands [...].” This implies an already enabled *Team*.
- CP-3301 mentions “[...] shared with [...] the stakeholders.” CP-3304 and CP-3307 imply *actively involved Stakeholders* that have to support the identification of conflicting requirements and at their prioritization.
- CP-3303 implies that the *value* of the *Opportunity* is already *established*.

Dependencies:

- depends on Opportunity::ValueEstablished (but dependency is already defined in predecessor Alpha State Bounded),
- depends on Team::Seeded, and
- depends on Stakeholders::Involved

B.O.3.4 *Alpha State Acceptable: The requirements describe a system that is acceptable to the stakeholders.*

Checkpoints:

1. The stakeholders accept that the requirements describe an acceptable solution. [3401]
2. The rate of change to the agreed requirements is relatively low and under control. [3402]
3. The value provided by implementing the requirements is clear. [3403]
4. The parts of the opportunity satisfied by the requirements are clear. [3404]
5. The requirements are testable. [3405]

Targeted by Activity Space(s):

- Shape the System(Stakeholders, Opportunity, Requirements, Software System, Work, Way-of-Working)

Analysis:

- CP-3401 implies that Stakeholders have to be actively involved. Does not yet imply that Stakeholders are in agreement (Stakeholders::InAgreement) since this State's checkpoints go beyond the conditions set by the checkpoints of CP-3401.
- CP-3403 and CP-3404 imply the value of the Opportunity is established, since this is needed to judge the "value provided [...]" and to identify "the parts of the Opportunity satisfied."

Dependencies:

- depends on Stakeholders::Involved (but dependency is already defined in predecessor Alpha State *Coherent*) and
- depends on Opportunity::ValueEstablished (but dependency is already defined in predecessor alpha state *Bounded*)

B.O.3.5 *Alpha State Addressed: Enough of the requirements have been addressed to satisfy the need for a new system in a way that is acceptable to the stakeholders.*

Checkpoints:

1. Enough of the requirements are addressed for the resulting system to be acceptable to the stakeholders. [3501]
2. The stakeholders accept the requirements as accurately reflecting what the system does and does not do. [3502]

3. The set of requirement items implemented provide clear value to the stakeholders. [3503]
4. The system implementing the requirements is accepted by the stakeholders as worth making operational. [3504]

Targeted by Activity Space(s):

- Test the System(Requirements, Software System, Way-of-Working)

Analysis:

- CP-3501 states “Enough of the requirements are addressed [...].” This implies a *Software System* that is *Ready*. The State *SoftwareSystem::Ready* includes a Checkpoint with the description “Operational support is in place.”, which is not necessarily precondition of this Alpha State.
- CP-3501 states “[...] acceptable to the stakeholders.” CP-3504 states “[...] accepted by the stakeholders as worth making operational.” This could imply that *Stakeholders* have to be *Satisfied For Deployment* BUT that State itself implies that *Requirements* have to be *Addressed* beforehand. Checkpoints CP-3501, CP-3502, and CP-3503 provide the state that is needed in order to let *Stakeholders* “[...] confirm that they agree that the system is ready for deployment”, hence they are preconditions to CP-1502. In turn, CP-1502 is regarded as a precondition to CP-3504 stating that “The system implementing the requirements *is accepted* by the stakeholders as worth making operational.” As the SEMAT Essence FAQ state, each “[...] checklist item should be considered from the context of the alpha it is a part of—taking a different perspective of the same phenomenon. While some checklist items seem closely related and possibly the same as a checklist item in another alpha, when you consider each alpha checklist from its own alpha perspective distinctions often become apparent.”[248] At this point, the semi-formal approach taken by SEMAT Essence offers a lot of flexibility that may result in ambiguities at times. While those ambiguities may not hinder pragmatic teams in their project work, they have to be resolved to provide a reasonable simulation model for the proposed simulation approach.

Dependencies:

- depends on *SoftwareSystem::Ready*
- to avoid a circular dependency further interdependencies have to be defined on Checkpoint level
 - CP-3504 depends on CP-1502 (*Stakeholders::SatisfiedForDeployment*)
 - CP-3501, CP-3502, and CP-3503 are preconditions to CP-1501 (*Stakeholders::SatisfiedForDeployment*)

B.O.3.6 *Alpha State Fulfilled: The requirements that have been addressed fully satisfy the need for a new system.*

Checkpoints:

1. The stakeholders accept the requirements as accurately capturing what they require to fully satisfy the need for a new system. [3601]
2. There are no outstanding requirement items preventing the system from being accepted as fully satisfying the requirements. [3602]
3. The system is accepted by the stakeholders as fully satisfying the requirements. [3603]

Targeted by Activity Space(s):

- Test the System(Requirements, Software System, Way-of-Working)

Analysis:

- With reaching this State all of the outstanding requirement items were implemented and accepted.

Dependencies:

- depends on SoftwareSystem::Ready (but dependency is already defined in predecessor Alpha State *Addressed*) and
- depends on Stakeholders::SatisfiedForDeployment (but dependency is already defined in predecessor Alpha State *Addressed*)

B.O.4 *Alpha: Software System*

B.O.4.1 *Alpha State Architecture Selected: An architecture has been selected that addresses the key technical risks and any applicable organizational constraints.*

Checkpoints:

1. The criteria to be used when selecting the architecture have been agreed on. [4101]
2. Hardware platforms have been identified. [4102]
3. Programming languages and technologies to be used have been selected. [4103]
4. System boundary is known. [4104]
5. Significant decisions about the organization of the system have been made. [4105]

6. Buy, build, and reuse decisions have been made. [4106]
7. Key technical risks agreed to. [4107]

Targeted by Activity Space(s):

- Shape the System(Stakeholders, Opportunity, Requirements, Software System, Work, Way-of-Working)

Analysis:

- The Checkpoints imply that “[...] most important usage scenarios for the system can be explained.” and to “[...] communicate the essential characteristics of the system [...].” These are Checkpoint descriptions of Requirements::Coherent.

Dependencies:

- depends on Requirements::Coherent

B.O.4.2 *Alpha State Demonstrable: An executable version of the system is available that demonstrates the architecture is fit for purpose and supports testing.*

Checkpoints:

1. Key architectural characteristics have been demonstrated. [4201]
2. The system can be exercised and its performance can be measured. [4202]
3. Critical hardware configurations have been demonstrated. [4203]
4. Critical interfaces have been demonstrated. [4204]
5. The integration with other existing systems has been demonstrated. [4205]
6. The relevant stakeholders agree that the demonstrated architecture is appropriate. [4206]

Targeted by Activity Space(s):

- Implement the System(Requirements, Software System, Way-of-Working), Test the System(Requirements, Software System, Way-of-Working)

Analysis:

- CP-4206 implies that *Stakeholders* are actively involved.

Dependencies:

- depends on Stakeholders::Involved

B.O.4.3 *Alpha State Usable: The system is usable and demonstrates all of the quality characteristics of an operational system.*

Checkpoints:

1. The system can be operated by stakeholders who use it. [4301]
2. The functionality provided by the system has been tested. [4302]
3. The performance of the system is acceptable to the stakeholders. [4303]
4. Defect levels are acceptable to the stakeholders. [4304]
5. The system is fully documented. Release content is known. [4305]
6. The added value provided by the system is clear. [4306]

Targeted by Activity Space(s):

- Implement the System(Requirements, Software System, Way-of-Working), Test the System(Requirements, Software System, Way-of-Working)

Analysis:

- CP-4301, CP-4303, and CP-4304 imply actively *involved Stakeholders*.
- The SEMAT Essence FAQ states that the last two Checkpoints could be rephrased to "The system is appropriately documented according to agreed documentation requirements." to reflect a more flexible description including agile and "crafted quality endeavors", where "documentation is often given less attention." [248]

Dependencies:

- depends on Stakeholders::Involved (but dependency is already defined in predecessor Alpha State *Demonstrable*)

B.O.4.4 *Alpha State Ready: The system (as a whole) has been accepted for deployment in a live environment.*

Checkpoints:

1. Installation and other user documentation are available. [4401]
2. The stakeholder representatives accept the system as fit-for-purpose. [4402]
3. The stakeholder representatives want to make the system operational. [4403]

4. Operational support is in place. [4404]

Targeted by Activity Space(s):

- Implement the System(Requirements, Software System, Way-of-Working), Test the System(Requirements, Software System, Way-of-Working)

Analysis:

- The wording of the Checkpoint descriptions of this Alpha State is ambiguous. Reading the descriptions of CP-4402 and CP-4403 could lead to the notion that *Stakeholders Alpha* would have to be in State *Satisfied for Deployment* beforehand. However, *Stakeholders* can only accept a *Software System* that was brought to proper shape. Thinking about cause and effect, the descriptions of both Checkpoints are interpreted here in a way that—IMHO—fits better to Checkpoints concerning the *Software System Alpha*. CP-4402 is read as “The Software System is in a condition/status/state/shape that Stakeholders (can) accept the system [...]” CP-4403 is interpreted as “The Software System is brought to a condition/status/state/shape that stakeholder representatives want to make the system operational.” This, in turn, lets Stakeholders “[...] confirm that they agree [...]” (CP-1502 of Stakeholders::SatisfiedForDeployment).

Dependencies:

- depends on none

B.O.4.5 *Alpha State Operational: The system is in use in an operational environment.*

Checkpoints:

1. The system has been made available to the stakeholders intended to use it. [4501]
2. At least one example of the system is fully operational. [4502]
3. The system is fully supported to the agreed service levels. [4503]

Targeted by Activity Space(s):

- Deploy the System(Stakeholders, Software System, Way-of-Working)

Analysis:

- The Alpha State and its Checkpoints imply that the Software System was deployed. This implies that *Stakeholders* were *Satisfied for Deployment* since the system would not get deployed otherwise.

Dependencies:

- depends on Stakeholders::SatisfiedForDeployment

B.O.4.6 *Alpha State Retired: The system is no longer supported.*

Checkpoints:

1. The system has been replaced or discontinued. [4601]
2. The system is no longer supported. [4602]
3. There are no “official” stakeholders who still use the system. [4603]
4. Updates to the system will no longer be produced. [4604]

Targeted by Activity Space(s):

- Operate the System(Stakeholders, Opportunity, Requirements, Software System, Way-of-Working)

Analysis:

- CP-4603 mentions “stakeholders” but does not define any conditions on them.

Dependencies:

- depends on none

B.O.5 *Alpha: Team*

B.O.5.1 *Alpha State Seeded: The team’s mission is clear and the know-how needed to grow the team is in place.*

Checkpoints:

1. The team mission has been defined in terms of the opportunities and outcomes. [5101]
2. Constraints on the team’s operation are known. [5102]
3. Mechanisms to grow the team are in place. [5103]
4. The composition of the team is defined. [5104]
5. Any constraints on where and how the work is carried out are defined. [5105]
6. The team’s responsibilities are outlined. [5106]
7. The level of team commitment is clear. [5107]
8. Required competencies are identified. [5108]
9. The team size is determined. [5109]
10. Governance rules are defined. [5110]

11. Leadership model is selected. [5111]

Targeted by Activity Space(s):

- Prepare to do the Work(Stakeholders, Opportunity, Requirements)

Analysis:

- This Alpha State already leads to a team consisting of a number of team members that is potentially already at the proper size—which may not be obvious in the first place. According to the SEMAT Essence FAQ Checkpoint CP-5103 could be rephrased to "Mechanisms to *sustain* and/or grow the team are in place." [248]
- CP-5101 implies that an *Opportunity* is already *Identified* and its *value* is already *established*.
- CP-5104 implies at least rough knowledge about the tasks at hand—that *Requirements* are *Bounded*.
- CP-5108 and CP-5109 imply that required competencies and the number of people needed are known. This again implies some knowledge about the things to accomplish.
- CP-5102..CP-5111 are interpreted as prerequisites to defining a *Way of Working*.

Dependencies:

- depends on Opportunity::ValueEstablished and
- depends on Requirements::Bounded

B.O.5.2 *Alpha State Formed: The team has been populated with enough committed people to start the mission.*

Checkpoints:

1. Individual responsibilities are understood. [5201]
2. Enough team members have been recruited to enable the work to progress. [5202]
3. Every team member understands how the team is organized and what their individual role is. [5203]
4. All team members understand how to perform their work. [5204]
5. The team members have met (perhaps virtually) and are beginning to get to know each other. [5205]
6. The team members understand their responsibilities and how they align with their competencies. [5206]

7. Team members are accepting work. [5207]
8. Any external collaborators (organizations, teams and individuals) are identified. [5208]
9. Team communication mechanisms have been defined. [5209]
10. Each team member commits to working on the team as defined. [5210]

Targeted by Activity Space(s):

- Coordinate Activity(Requirements, Team, Work, Way of Working)

Analysis:

- CP-5201 and CP-5204 imply that *foundations* of a *Way of Working* are *established*. Otherwise “individual responsibilities” and “how to perform work” could not be understood.
- CP-5207 states “Team members are accepting work.” This could imply that *Work* should be *Prepared* before (“[...] tasks identified and prioritized [...] broken down sufficiently [...] cost and effort estimated [...]”) because otherwise there would be nothing (concrete) to accept. On the other hand *Work::Prepared* (CP-6201) demands that “Commitment is made.” CP-5207 is interpreted here as “Team members are willing and able to accept work.” This fits well with Alpha State’s description “been populated with enough committed people to start the mission.”
- CP-5208 implies that *Stakeholders* are already *Recognized*.

Dependencies:

- depends on Stakeholders::Recognized and
- depends on WayOfWorking::FoundationEstablished

B.O.5.3 *Alpha State Collaborating: The team members are working together as one unit.*

Checkpoints:

1. The team is working as one cohesive unit. [5301]
2. Communication within the team is open and honest. [5302]
3. The team is focused on achieving the team mission. [5303]
4. The team members know each other. [5304]

Targeted by Activity Space(s):

- Support the Team(Team, Work, Way of Working)

Analysis:

- CP-5301 states the “[...] team is working [...].” This implies that *Work* already had been *Started*.

Dependencies:

- depends on Work::Started

B.o.5.4 *Alpha State Performing: The team is working effectively and efficiently.*

Checkpoints:

1. The team consistently meets its commitments. [5401]
2. The team continuously adapts to the changing context. [5402]
3. The team identifies and addresses problems without outside help. [5403]
4. Effective progress is being achieved with minimal avoidable backtracking and reworking. [5404]
5. Wasted work, and the potential for wasted work are continuously eliminated. [5405]

Targeted by Activity Space(s):

- Track Progress(Requirements, Team, Work, Way of Working)

Analysis:

- CP-5401 may raise the question “How does the team consistently meet its commitments?” Here is assumed that this gets accomplished by “[...] using and adapting the way-of-working to suit their current context.” (CP-7501 of WayOfWorking::WorkingWell) In author’s opinion, this fits well with values and attitudes to facilitate in SE courses and is corroborated by CP-5402..CP-5405.

Dependencies:

- depends on WayOfWorking::WorkingWell

B.o.5.5 *Alpha State Adjourned: The team is no longer accountable for carrying out its mission.*

Checkpoints:

1. The team responsibilities have been handed over or fulfilled. [5501]

2. The team members are available for assignment to other teams. [5503]
3. No further effort is being put in by the team to complete the mission. [5503]

Targeted by Activity Space(s):

- *Stop the Work(Requirements, Team, Work, Way of Working)*

Analysis:

- Checkpoints do not indicate any requirements on other Alpha States.

Dependencies:

- depends on none

B.0.6 *Alpha: Work*

B.0.6.1 *Alpha State Initiated: The work has been requested.*

Checkpoints:

1. The result required of the work being initiated is clear. [6101]
2. Any constraints on the work's performance are clearly identified.[6102]
3. The stakeholders that will fund the work are known. [6103]
4. The initiator of the work is clearly identified. [6104]
5. The stakeholders that will accept the results are known. [6105]
6. The source of funding is clear. [6106]
7. The priority of the work is clear. [6107]

Targeted by Activity Space(s):

- Prepare to do the Work(Stakeholders, Opportunity, Requirements)

Analysis:

- CP-6101 states that "The result of the work initiated is clear." This implies that the *value* of the *Opportunity* has been *established* and that *Requirements* are at least *Bounded*.
- CP-6103..CP-6106 imply that *Stakeholders* are *Recognized*.
- CP-6107 implies that *Requirements* are *Coherent* since they could hardly be prioritized reasonably.

Dependencies:

- depends on Opportunity::ValueEstablished,
- depends on Requirements::Coherent, and
- depends on Stakeholders::Recognized

B.O.6.2 *Alpha State Prepared: All pre-conditions for starting the work have been met.*

Checkpoints:

1. Commitment is made. [6201]
2. Cost and effort of the work are estimated. [6202]
3. Resource availability is understood. [6203]
4. Governance policies and procedures are clear. [6204]
5. Risk exposure is understood. [6205]
6. Acceptance criteria are defined and agreed with client. [6206]
7. The work is broken down sufficiently for productive work to start. [6207]
8. Tasks have been identified and prioritized by the team and stakeholders. [6208]
9. A credible plan is in place. [6209]
10. Funding to start the work is in place. [6210]
11. The team or at least some of the team members are ready to start the work. [6211]
12. Integration and delivery points are defined. [6212]

Targeted by Activity Space(s):

- Prepare to do the Work(Stakeholders, Opportunity, Requirements)

Analysis:

- CP-6201 implies that a *Team* is *Seeded* and *Stakeholders* are *Represented* to make a commitment.
- CP-6206, CP-6208, and CP-6210 imply an already *Seeded Team*, *Represented* as well as *Involved Stakeholders*, and *Coherent Requirements*.
- CP-6211 implies an already *Seeded Team*.
- CP-6205's implications keep unclear to some extent since "risk exposure" is a rather broad term.
- CP-6212 could imply a *Software System* with a *selected architecture*.

Dependencies:

- depends on Team::Seeded,
- depends on Stakeholders::Involved, and
- depends on Requirements::Coherent (but dependency is already defined in predecessor Alpha State *Initiated*)

B.o.6.3 *Alpha State Started: The work is proceeding.*

Checkpoints:

1. Development work has been started. [6301]
2. Work progress is monitored. [6302]
3. The work is being broken down into actionable work items with clear definitions of done. [6303]
4. Team members are accepting and progressing tasks. [6304]

Targeted by Activity Space(s):

- Coordinate Activity(Requirements, Team, Work, Way of Working)

Analysis:

- CP-6304 implies an already active *Team*.

Dependencies:

- depends on Team::Seeded (but dependency is already defined in predecessor Alpha State *Prepared*)

B.o.6.4 *Alpha State Under Control: The work is going well, risks are under control, and productivity levels are sufficient to achieve a satisfactory result.*

Checkpoints:

1. Tasks are being completed. [6401]
2. Unplanned work is under control. [6402]
3. Risks are under control as the impact if they occur and the likelihood of them occurring have been reduced to acceptable levels. [6403]
4. Estimates are revised to reflect the team's performance. [6404]
5. Measures are available to show progress and velocity. [6405]
6. Re-work is under control. [6406]
7. Tasks are consistently completed on time and within their estimates. [6407]

Targeted by Activity Space(s):

- Coordinate Activity(Requirements, Team, Work, Way of Working),
- Track Progress(Requirements, Team, Work, Way of Working)

Analysis:

- CP-6404 mentions the performance of a team. CP-6407 implies a *Team* that is *Performing*. Another perspective could be the *Team* is *Performing* because of the *Work* that is *Under Control*. With the chosen dependency it is emphasized that a performing team is needed to get work under control.

Dependencies:

- depends on Team::Performing

B.o.6.5 *Alpha State Concluded: The work to produce the results has been concluded.*

Checkpoints:

1. All outstanding tasks are administrative housekeeping or related to preparing the next piece of work. [6501]
2. Work results have been achieved. [6502]
3. The stakeholder(s) has accepted the resulting software system. [6503]

Targeted by Activity Space(s):

- Track Progress(Requirements, Team, Work, Way of Working)

Analysis:

- CP-6502 is formulated in a very general manner. It is assumed that Requirements got Fulfilled.
- CP-6503 states “[...] stakeholder(s) has accepted the resulting software system.” There is no hint to the deployment of the system. Hence it is assumed that *Stakeholders* are not just *SatisfiedForDeployment* but *SatisfiedInUse*.

Dependencies:

- depends on Requirements::Fulfilled and
- depends on Stakeholders::SatisfiedInUse

B.o.6.6 *Alpha State Closed: All remaining housekeeping tasks have been completed and the work has been officially closed.*

Checkpoints:

1. Lessons learned have been itemized, recorded and discussed. [6601]
2. Metrics have been made available. [6602]

3. Everything has been archived. [6603]
4. The budget has been reconciled and closed. [6604]
5. The team has been released. [6605]
6. There are no outstanding, uncompleted tasks. [6606]

Targeted by Activity Space(s):

- Stop the Work(Requirements, Team, Work, Way of Working)

Analysis:

- CP-6605 mentions an already released *Team*.

Dependencies:

- depends on Team::Adjourned

B.0.7 *Alpha: Way of Working*

B.0.7.1 *Alpha State Principles Established: The principles, and constraints, that shape the way-of-working are established.*

Checkpoints:

1. Principles and constraints are committed to by the team. [7101]
2. Principles and constraints are agreed to by the stakeholders. [7102]
3. The tool needs of the work and its stakeholders are agreed. [7103]
4. A recommendation for the approach to be taken is available. [7104]
5. The context within which the team will operate is understood. [7105]
6. The constraints that apply to the selection, acquisition, and use of practices and tools are known. [7106]

Targeted by Activity Space(s):

- Prepare to do the Work(Stakeholders, Opportunity, Requirements)

Analysis:

- CP-7101 implies active team members are committing.

- CP-7102 and CP-7103 indicate active stakeholders to agree. Defining *Stakeholders::Represented* as a precondition to this Alpha State would introduce an indirect circular dependency since *Stakeholders::Represented* (CP-1204) requires this State's successor *WayOfWorking::FoundationEstablished* as a dependency. To avoid a circular dependency, they have to be defined on the Checkpoint level. In a real SE endeavor, this would represent the process of recognizing, choosing and recommending a *Way Of Working*, getting agreement from *Stakeholders* and eventually using it.

Dependencies:

- depends on Team::Seeded,
- to avoid an indirect circular dependency, further interdependencies have to be defined on Checkpoint level
 - depends on CP-1201 (Stakeholders::Represented),
 - depends on CP-1202 (Stakeholders::Represented), and
 - depends on CP-1203 (Stakeholders::Represented)

B.O.7.2 *Alpha State Foundation Established: The key practices, and tools, that form the foundation of the way of working are selected and ready for use.*

Checkpoints:

1. The key practices and tools that form the foundation of the way-of-working are selected. [7201]
2. Enough practices for work to start are agreed to by the team. [7202]
3. All non-negotiable practices and tools have been identified. [7203]
4. The gaps that exist between the practices and tools that are needed and the practices and tools that are available have been analyzed and understood. [7204]
5. The capability gaps that exist between what is needed to execute the desired way of working and the capability levels of the team have been analyzed and understood. [7205]
6. The selected practices and tools have been integrated to form a usable way-of-working. [7206]

Targeted by Activity Space(s):

- Prepare to do the Work(Stakeholders, Opportunity, Requirements)

Analysis:

- CP-7202 and CP-7205 mention the “team”. This implies an already existing *Team*.

Dependencies:

- depends on Team::Seeded (but dependency is already defined in predecessor Alpha State *Principles Established*)

B.O.7.3 *Alpha State In Use: Some members of the team are using, and adapting, the way-of-working.*

Checkpoints:

1. The practices and tools are being used to do real work. [7301]
2. The use of the practices and tools selected are regularly inspected. [7302]
3. The practices and tools are being adapted to the team’s context. [7303]
4. The use of the practices and tools is supported by the team. [7304]
5. Procedures are in place to handle feedback on the team’s way of working. [7305]
6. The practices and tools support team communication and collaboration. [7306]

Targeted by Activity Space(s):

- Support the Team(Team, Work, Way of Working)

Analysis:

- CP-7301 mentions “real work” implying that *Work* has already been *Started*.
- CP-7304 mentions “[...] supported by the team.” This implies an already active *Team*.

Dependencies:

- depends on Work::Started and
- depends on Team::Formed

B.O.7.4 *Alpha State In Place: All team members are using the way of working to accomplish their work.*

Checkpoints:

1. The practices and tools are being used by the whole team to perform their work. [7401]
2. All team members have access to the practices and tools required to do their work. [7402]
3. The whole team is involved in the inspection and adaptation of the way-of-working. [7403]

Targeted by Activity Space(s):

- Support the Team(Team, Work, Way of Working)

Analysis:

- CP-7401 states “[...] used by the *whole* team [...].” CP-7402 states “All team members [...] to do their work.” Both imply a *Team* that is already *Formed*.

Dependencies:

- depends on Team::Formed (but dependency is already defined in predecessor Alpha State *In Use*)

B.O.7.5 *Alpha State Working well: The team’s way of working is working well for the team.*

Checkpoints:

1. Team members are making progress as planned by using and adapting the way-of-working to suit their current context. [7501]
2. The team naturally applies the practices without thinking about them. [7502]
3. The tools naturally support the way that the team works. [7503]
4. The team continually tunes their use of the practices and tools. [7504]

Targeted by Activity Space(s):

- Track Progress(Requirements, Team, Work, Way of Working)

Analysis:

- The three Alphas of the *Endeavor* area of concern describe the *Team* and how it accomplishes its *Work*. Naturally, these three alphas are very interconnected. The endeavor will be progressing ideally if the *Team* is in state *Performing*, *Work* is in state *Under Control*, and the *Way of Working* is in state *Working Well*. Since the *Way of Working* is guiding the *Work* Alpha and the *Team* is applying the *Way of Working*¹, the latter *Alpha* is seen as a precondition to the other two Alphas—not the other way around. Other perspectives may be valid as well.

Dependencies:

- no further dependencies

B.O.7.6 *Alpha State Retired: The way of working is no longer in use by the team.*

Checkpoints:

1. The team's way of working is no longer being used. [7601]
2. Lessons learned are shared for future use. [7602]

Targeted by Activity Space(s):

- Stop the Work(Requirements, Team, Work, Way of Working)

Analysis:

- CP-7601 implies that Team's *Way of Working* is not longer used and needed when no more *Work* has to be done—when there is nothing left that the *Way of Working* could guide.

Dependencies:

- depends on Work::Closed

B.1 ALIGNMENT OF ANALYSIS RESULTS

By adding all defined dependencies into the dependency graph described in section 3.3 on page 120 it results in a graph that is presented in figure 3.7 on page 130. The resulting dependency graph contains no cycles. Hence a resulting simulation model is executable in a way that every Checkpoint can be fulfilled. Hence every Alpha State can be reached given that each individual effort x , measured in person hours, associated with one of the Checkpoints in the simulation model is $x \in \mathbb{N}$, $0 \leq x < \infty$.

Taking the explicitly defined entry criteria of Activity Spaces, introduced in version 1.1 of the Essence kernel[188], into account, no inconsistencies are emerging.

¹ This is defined by the Alpha Associations of the Essence kernel.

HEURISTICS

This chapter provides mappings of the *Integrated Approach* and its components to

- quality criteria defined by an quality assessment framework[74]
- recommendations based on the synthesis of a comprehensive SLR[126] (cf. section C.3 on page 366), as well as
- SE education requirements based on conclusions of Boehm[27] reviewing SE in the 20th and 21st century (cf. section C.2 on page 366)

to enable the assessment of their contribution to the stated learning objectives (cf. section 4.1.3.1 on page 186) and current SE education in general.

C.1 DONDI AND MORETTI (2007) QUALITY CRITERIA AND THEIR MAPPING TO SIM4SEED'S APPROACHES AND DESIGN DECISIONS

SIG-GLUE QUALITY CRITERIA	SIM4SEED APPROACH
<i>Pedagogical and context criteria</i>	
Target groups and prerequisites	
Identification of target groups	target groups of learners and lecturers were analyzed and described (see section 4.1.3)
Identification of prerequisites	prerequisites were analyzed and described (see section 4.1.3)
Learning objectives	
Clear definition of objectives	learning objectives were analyzed and described (see section 4.1.3)
Correspondence between established objectives and the objectives that can actually be reached by using the learning game	given by the design of the provided games for that specific purpose
Context of usage	
Clarity of practical instructions for the use of the learning game	given in introduction

SIG-GLUE QUALITY CRITERIA	SIM4SEED APPROACH
Indications/suggestions on the context in which the learning game can be used	different usage scenarios described (cf. section 4.4)
Coherence of the game with the targeted context	designed specifically for this purpose
Coherence between the learning game structure and the planned training and learning context	designed specifically for this purpose
Link between the learning game activities and the professional/working context	<i>Integrated Approach</i> designed specifically for this purpose (cf. chapter 4)
Didactic strategy	
Indication of the average play time	described in detail (cf. section 5.1)
Incentives and support to motivation	Simulation Game strongly oriented towards intrinsic motivation (cf. section 4.7 on page 226)
Support to engagement and fun	Simulation Game utilizing primarily intrinsic characteristics of the domain and tools provided (4.7 on page 226)
Coherence between the game strategy and learning objectives	games designed specifically for this purpose (cf. sections 4.5 on page 207 and 4.7 on page 226)
Quality of the game strategy with the individual player characteristics	supporting different types of players
Clarity of the game environment/setting	provided in introduction
Organisation and structure of the learning game	oriented towards typical utilization of concepts and tools in real SE project work
Clarity of the rules to be followed and decision making process	oriented towards typical utilization of concepts and tools in real SE project work
Coherence between rules and consequence	oriented towards typical utilization of concepts and tools in real SE project work
Constant focus on the player experience	providing immediate feedback (of individual as well as own team's and other teams' performance)
Clear definition of roles (e.g., player, instructors, animators, etc)	different perspectives provided for players and lecturers

SIG-GLUE QUALITY CRITERIA	SIM4SEED APPROACH
Coherence of the social and collaborative activity with the objectives	oriented towards social constructivism, by combining individual game play with collaboration approaches in teams
Communication and media	
Clear and user-friendly tone and language	given in <i>Essence Kernel Puzzler</i> and <i>Simulation Game</i> , both specifically designed for this target group
Quality of the interaction between the learning game and the user/player	both <i>Essence Kernel Puzzler</i> and <i>Simulation Game</i> providing immediate feedback, interaction in <i>Simulation Game</i> oriented towards requirements of utilization of concepts in real project work
Quality of the interaction among users/players, etc.	facilitated by triggers for social interaction, discussion and reflection
Coherence between the media used in the learning game and the contents, the established objectives and the target group	providing tools to be used inside of the <i>Simulation Game</i> and real project work
Evaluation	
Clear identification of evaluation criteria and procedures	cf. section 5 on page 251
Adequate number and distribution of evaluation activity, during the game and at the end	cf. section 5 on page 251
Type of evaluation activity proposed	cf. section 5 on page 251
Quality of the feedback to the evaluation activity	cf. section 5 on page 251
Relevance of evaluation activity and consistency with the objectives and/or the contents	cf. section 5 on page 251
Support to the reflexive process (e.g., players can review and rethink their performance)	specifically addressed by collaborative approach and <i>Time Travel</i> feature of the <i>Simulation Game</i>
Content criteria	
Correct technical/scientific language and contents	given by utilizing underlying domain (SEMAT Essence)

SIG-GLUE QUALITY CRITERIA	SIM4SEED APPROACH
Updating or obsolescence of contents	given by utilizing underlying domain (SEMAT Essence)
Correct and logical organisation of contents	given by utilizing underlying domain (SEMAT Essence)
Link between the contents and the subject area/knowledge domain/curriculum	given by utilizing underlying domain (SEMAT Essence) (cf. section 4.2)
Practical contextualisation of the content	designed specifically to prepare for real project work
Correct balance of the context in relation with the target group	addressed by the <i>Integrated Approach</i> specifically designed for this purpose
Coherence of contents with the established objectives and the target group	addressed by the <i>Integrated Approach</i> specifically designed for this purpose
<i>Technical criteria</i>	
Credits	
Information on the producers, authors, etc	omitted in case study's deployment, as author/producer was introduced and on site
Portability and conformance to standards	
Robustness of the game	utilizing today's web standards
Conformance to standard	utilizing today's web standards
Structure and organisation	
Easy to be installed (for off-line digital games)	online game, delivered to current web browsers
Modularity of the design	cf. section
Modularity in the use	provided by phases of the <i>Integrated Approach</i> (cf. chapter 4)
Aesthetic and usage of the media	
Quality of user/game interface	functional UI, reduced to necessary elements, avoiding extrinsic cognitive load
Possibility of intervention on the use of materials (stop, rewind)	<i>Essence Kernel Puzzler</i> : levels can be repeated for an unlimited number of times, <i>Simulation Game</i> : provided by the <i>Time Travel</i> feature

SIG-GLUE QUALITY CRITERIA	SIM4SEED APPROACH
Positioning of the different elements on the screen	following accepted standards of today's web design
Technical quality	
Quality of image definition	utilizing scalable vector formats
Rhythm of images	reduced to the necessary minimum, avoiding extrinsic cognitive load
Quality and definition of audio	no audio provided
Integration between audio and image elements	no audio provided
Synchronism between audio and image elements	no audio provided
Quality of typographic characteristics and clarity of texts	following accepted standards of today's web and mobile design
Quality of image composition	following accepted standards of today's web design, reduced to necessary minimum, avoiding extrinsic cognitive load
Technical quality of drawings	
Technical quality of pictures	
Technical quality of graphic animations	
Information produced	
Privacy and security of personal data	provided by standard procedures of today's transport encryption and approved security standards used in web development
Storage of the gaming time	part of embedded assessment (in-game measurements)
Storage of evaluation and activities results (e.g., save progress)	game state is persisted
Print of the information	enabled by web browser, basically no need to print

Table C.2: Special Interest Group for Game-Based Learning in Universities and Lifelong Learning Quality Criteria[74] and Their Mapping to Taken Approaches and Design Decisions

C.2 EVALUATING MAPPING TO SE EDUCATION REQUIREMENTS BY BOEHM (2006)

Section 1.2 on page 5 provided conclusions of Boehm[27] reviewing SE in the 20th and 21st century. With regards to his conclusions, the *Integrated Approach* contributes to the formulated requirements:

- The approach considers a move towards flexible SE practices (cf. section 2.6.1 on page 88). Hence it is “Anticipating future trends and preparing students to deal with them.”[27]
- By utilizing SEMAT Essence and its kernel (cf. section 2.7 on page 95), it is “separating timeless principles from out-of-date practices.”[27]
- With its combination of learning games preparing students for their course project work, it is aiming at “Packaging smaller-scale educational experiences in ways that apply to large-scale projects.”[27]
- By utilizing an emerging innovative standard it is “Participating in leading-edge software engineering research and practice, and incorporating the results into the curriculum.”[27]
- By utilizing simulation and DGBL it is “Helping students learn how to learn, through [...] future-oriented educational games and exercises [...]”

C.3 EVALUATING MAPPING TO RECOMMENDATIONS OF JIANG ET AL. (2015)

As synthesis of their comprehensive and rigorous SLR Jiang et al.[126] provide a list of recommendations for the implementation of simulation- and game-based learning activities in SE education. Table C.3 on the facing page summarizes the mapping of their recommendations to the game-based learning approaches taken in the case study described in this thesis. It can be summarized that the game-based approaches taken in the case study follow recommendations given, except for the mandatory assignment. The preparation of pre-instructions as well as the provision of right examples, e.g. by providing an introductory tutorial in the *Simulation Game* are considered for future work.

Recommendation	Consideration in the Integrated Approach
Allocate proper playing time for specific simulation.	The time needed to play the <i>Simulation Game</i> is limited by intention. Students were given the opportunity to continue to play the <i>Simulation Game</i> outside of the regular course meetings. Students perceived the duration of the game as appropriate (cf. figure 5.15 on page 270). Time allocation to play the <i>Puzzler</i> was left to the students themselves.
Make mandatory assignment.	This approach was not chosen in the case study but may be examined in future work.
Add collaborative aspects.	Players of the <i>Simulation Game</i> are members of teams to foster collaboration (cf. section 4.7.5 on page 239).
Add competitive aspects.	Teams compete in the <i>Simulation Game</i> for best team score (cf. section 4.7.5 on page 239).
Teach students relevant theoretical knowledge.	<i>Phase 1</i> of the <i>Integrated Approach</i> prepared students for the <i>Simulation Game</i> . Motivation, basic concepts, elements and relationships were introduced by an educational case study and the <i>Puzzler</i> game.
Prepare pre-instructions well.	Pre-instructions were provided live at conducting the case study. The evaluation of the questionnaire revealed that student wish some kind of introductory tutorial, which is part of future work.
Show students right examples.	The interface of the <i>Simulation Game</i> and of the <i>Essence Navigator</i> were demonstrated live, right before the start of the game. The evaluation of the questionnaire revealed that student wish some kind of introductory tutorial, which is part of future work.
Encourage planning before playing.	Planning before acting in the <i>Simulation Game</i> gets rewarded by less fees raised for time travelling.
Encourage the attempts of different playing approaches.	Observations and in-game measurements showed different playing approaches (cf. section 5.2.2.2 on page 262).

Table C.3: Evaluating Mapping of Approaches Taken to Recommendations of Jiang et al.[126]

STATISTICAL HYPOTHESES TESTING

This appendix provides an exemplary procedure for hypothesis testing utilized in the statistical analysis of case study results.

Having only a small set of observations ($n < 20$) without any knowledge about the particular distribution of the data only a *non-parametric* statistical test, able to handle small sets of observations, is qualified for this study. The *Sign-test* fulfilling these criteria was chosen. It is a non-parametric binomial test about the median η of a population.

D.1 HYPOTHESIS TEST FOR Q09

Observations are at ordinal scale {1,2,3,4,5} with 1 representing a value with the minimum agreement, 3 representing a neutral value and 5 representing the value with the maximum agreement.

Step 1: Hypothesis Statement

$H_0 : \eta_0 \leq 3$	$H_1 : \eta_0 > 3$
students DO NOT	students DO perceive
perceive fun at playing	fun at playing the
the simulation game	simulation game
$\alpha = .05$	

Step 2: Test Statistic

S_+ number of observations $x_i > \eta_0 =$	11
S_- number of observations $x_i < \eta_0 =$	0
S_0 number of observations $x_i = \eta_0 =$	8
after S_0 observations are eliminated, $n =$	11

Step 3: P-value

X is a binomial random variable representing “the number of observations greater than $\eta_0 = 3$ ”, S_+ is the observed value of X , η_0 represents the (hypothetical) median, $P(X < \eta) = P(X > \eta) = 1/2 = .5$

$$\begin{aligned}
 Pval &= P(X \geq 11) \\
 &= 1 - P(X \leq 10) \\
 &= 1 - b(n, S_+ - 1, p) = 1 - b(11, 10, .5) \\
 &= 1 - 1 \\
 &= 0
 \end{aligned}$$

Step 4: Decision

$Pval = 0 < \alpha = .05$, we reject H_0 and accept H_1 .

Step 5: Conclusion

The data provide sufficient evidence to conclude at $\alpha = .05$ that η is greater than 3. Hence students DO perceive fun at playing the simulation game.

D.2 HYPOTHESIS TEST FOR Q19

Observations are at ordinal scale {1,2,3,4,5} with 1 representing a value with the minimum agreement, 3 representing a neutral value and 5 representing the value with the maximum agreement.

Step 1: Hypothesis Statement

$H_0 : \eta_0 \leq 3$	$H_1 : \eta_0 > 3$
students would NOT	students WOULD
recommend the	recommend the
simulation game to a	simulation game to a
friend or fellow student	friend or fellow student
$\alpha = .05$	

Step 2: Test Statistic

S_+ number of observations $x_i > \eta_0 =$	15
S_- number of observations $x_i < \eta_0 =$	0
S_0 number of observations $x_i = \eta_0 =$	4
after S_0 observations are eliminated, $n =$	15

Step 3: P-value

X is a binomial random variable representing “the number of observations greater than $\eta_0 = 3$ ”, S_+ is the observed value of X , η_0 represents the (hypothetical) median, $P(X < \eta) = P(X > \eta) = 1/2 = .5$

$$\begin{aligned}
 Pval &= P(X \geq 15) \\
 &= 1 - P(X \leq 14) \\
 &= 1 - b(n, S_+ - 1, p) = 1 - b(15, 14, .5) \\
 &= 1 - 1 \\
 &= 0
 \end{aligned}$$

Step 4: Decision

$Pval = 0 < \alpha = .05$, we reject H_0 and accept H_1 .

Step 5: Conclusion

The data provide sufficient evidence to conclude at $\alpha = .05$ that η is greater than 3. Hence students WOULD recommend the simulation game to a friend or fellow student.

D.3 HYPOTHESES TESTS

Following the schema of presented hypothesis tests, the remaining ones are presented in short form summarized in the following table D.1.

	S_+	S_-	S_0	n	$Pval$	Accepted	Rejected
Q09	11	0	8	11	0	H_1	H_0
Q19	15	0	4	15	0	H_1	H_0
Q20	18	0	1	18	0	H_1	H_0
Q22	18	1	0	19	0	H_1	H_0
Q23	15	2	2	17	.001	H_1	H_0
Q25	11	2	6	13	.011	H_1	H_0
Q27	14	1	4	15	0	H_1	H_0
Q33	9	1	3	10	.011	H_1	H_0
Q35	9	1	3	10	.011	H_1	H_0
Q40	15	0	4	15	0	H_1	H_0
Q42	1	7	11	8	.996	H_0	H_1
Q43	18	0	1	18	0	H_1	H_0
Q44	14	0	5	14	0	H_0	H_1
Q45	13	0	6	13	0	H_0	H_1
S_+ number of observations $x_i >$ η_0, S_- number of observations $x_i <$ η_0, S_0 number of observations $x_i = \eta_0, \alpha = .05, H_0 : \eta_0 \leq 3, H_1 :$ $\eta_0 > 3$							

Table D.1: Hypothesis Tests Summary

QUESTIONNAIRE USED FOR CASE STUDY

This chapter provides the questionnaire used for the case study conducted. The questionnaire results were collected digitally using an online platform.

HINTERGRUND

1. Q01: Spielen Sie digitale Spiele? (Skala 1-5: 1=nie, 5=regelmäßig)
2. Q02: Welche Spielplattformen nutzen Sie? (Auswahl: „PC“, „Web-Browser“, „Spielkonsole“, „Smartphone“, „Tablet“, „Offline (nicht digital)“, Skala 1-3: 1=nie, 2=selten, 3=häufig)
3. Q03: Wie viel Zeit verbringen Sie wöchentlich mit digitalen Spielen (egal auf welcher Plattform)? (Auswahl: <1h, 1h-3h, 3h-5h, 5h-10h, >10h)
4. Q04: Welches Spielgenre bevorzugen Sie? (Auswahl: „Adventures“, „Actionspiele“, „Beat’ em up“, „Ego/Third Person Shooter“, „Geschicklichkeitsspiele“, „Jump ‘n’ Run“, „Open World Spiele“, „Puzzle/Quiz Spiele“, „Rollenspiele“, „Sportspiele“, „Strategiespiele“, „Simulationsspiele“, „Aufbausimulationen“, „Sonstige“)
5. Q05: Welche Spiele-Genres bevorzugen Sie über die Auswahl oben hinaus?
6. Q06: Meinen Sie Spielen und Lernen gehören zusammen? (Skala 1-5: 1=„gar nicht“, 5=„unbedingt“)
7. Q07: Haben Sie bereits an Softwareprojekten außerhalb der SE-Lehrveranstaltungen mitgewirkt (z.B. im Praxissemester, vor/-neben dem Studium)? (Auswahl: „Ja“/„Nein“)
8. Q08: Was bevorzugen Sie zum Lernen von neuen Software Engineering (SE) Konzepten/Vokabular? Geben Sie bitte eine Reihenfolge Ihrer Präferenzen an (mehrfache Belegung eines Platzes ist erlaubt). (Auswahl: „Vorlesung“, „Spiel“, „eigenes Literaturstudium“, „Fallstudien“, „eigenes Softwareprojekt“, „Kombination der Lernformen“, Ranking: 1-6)

SPIEL / SPIELAUFBAU / SPIELVERLAUF

9. Q09: Wie viel Freude hat Ihnen das Spiel bereitet? (Skala 1-5: 1="gar keine", 5="sehr viel")
10. Q10: Wie leicht oder schwer fanden Sie das Spiel? (Skala 1-5: 1="sehr leicht", 5="sehr schwer")
11. Q11: Wie fanden Sie die Dauer des Spiels? (Auswahl: „zu kurz“, „angemessen“, „zu lang“)
12. Q12: Welcher Aspekt/Teil des Spiels gefiel Ihnen am besten? Warum?
13. Q13: Welcher Aspekt/Teil des Spiels gefiel Ihnen am wenigsten? Warum?
14. Q14: Gab es im Spiel Faktoren, die Sie gestört oder verwirrt haben? (Auswahl: „Ja“/„Nein“)
15. Q15: Welche Faktoren haben Sie gestört und/oder verwirrt?
16. Q16: Was würden Sie an diesem Spiel verändern, um es noch besser zu machen?
17. Q17: Werden Sie das Spiel noch einmal spielen? (Auswahl: „Ja“/„Nein“)
18. Q18: Warum werden Sie das Spiel (nicht) noch einmal spielen?
19. Q19: Würden Sie das Spiel einem Kommilitonen/Freund empfehlen? (Skala 1-5: 1="gar nicht", 5="unbedingt")
20. Q20: Sollte das Spiel standardmäßig Bestandteil einer SE-Lehrveranstaltung sein?(Skala 1-5: 1="gar nicht", 5="unbedingt")
21. Q21: Warum sollte das Spiel standardmäßig Bestandteil einer SE-Lehrveranstaltung sein?

LERNEFFEKT

22. Q22: Haben Sie den Eindruck, im Spiel insgesamt etwas über SE gelernt zu haben? (Skala 1-5: 1="gar nicht", 5="unbedingt")
23. Q23: Haben Sie den Eindruck, mit dem Spiel Wissen aus den Software Engineering Veranstaltungen aufgefrischt zu haben? (Skala 1-5: 1="gar nicht", 5="unbedingt")
24. Q24: Warum haben Sie den Eindruck, mit dem Spiel (kein) Wissen aufgefrischt zu haben?
25. Q25: Haben Sie den Eindruck, mit dem Spiel etwas gelernt zu haben, was Sie nicht (so gut) zuvor in SE-Lehrveranstaltungen gelernt haben? (Skala 1-5: 1="gar nicht", 5="unbedingt")

26. Q26: Was haben Sie im Spiel gelernt, was Sie zuvor nicht (so gut) in SE-Lehrveranstaltungen lernen konnten?
27. Q27: Haben Sie den Eindruck, sich im Verlauf des Spiels verbessert zu haben (höhere Punktzahl bei gleichen Kosten)? (Skala 1-5: 1="gar nicht", 5="unbedingt")
28. Q28: Was hat Ihnen dabei am meisten geholfen? (Skala: 1="sehr stark", 2="weniger stark", 3="gar nicht" / Auswahl: "Kommunikation im Team", "Möglichkeit in der Zeit zu reisen", "Ranking/Leaderboards", "Essence Navigator")
29. Q29: Was hat Ihnen am meisten dabei geholfen, aus dem Spiel zu lernen? (Skala: 1="sehr stark", 2="weniger stark", 3="gar nicht" / Auswahl: "Kommunikation im Team", "Möglichkeit in der Zeit zu reisen", "Ranking/Leaderboards", "Essence Navigator", "Auswertung in der ganzen Gruppe", "selbst aktiv über Entscheidungen nachdenken zu müssen", "Spaß am Spiel")
30. Q30: Hat Sie die englische Sprache im Spiel gestört?(Skala 1-5: 1="gar nicht", 5="sehr stark")

ESSENCE KERNEL PUZZLER

31. Q31: Haben Sie in Vorbereitung auf das Spiel den Essence Kernel Puzzler genutzt? (Auswahl: „Ja“/„Nein“)
Antworten Sie auf die nun folgenden Fragen bitte nur, falls Sie den Essence Kernel Puzzler genutzt haben.
32. Q32: Wie viel Zeit haben Sie mit dem Puzzler insgesamt verbracht? (Auswahl: <10min, 10min-20min, 20-30min, 30-60min, >60min)
33. Q33: Wie sehr hat Ihnen der Puzzler geholfen, sich mit Essence Vokabular und Konzepten vertraut zu machen? (Skala 1-5: 1="gar nicht", 5="sehr stark")
34. Q34: Wie schwer fanden Sie den Puzzler insgesamt (3 = angemessen)? (Skala 1-5: 1="zu leicht", 3="angemessen", 5="zu lang")
35. Q35: Werden Sie den Puzzler wieder benutzen, um Essence Vokabular aufzufrischen? (Skala 1-5: 1="sicher nicht", 5="sehr wahrscheinlich")
36. Q36: Hat Sie der Puzzler gut auf das Simulationsspiel vorbereitet? (Auswahl: „Ja“/„Nein“)
37. Q37: Werden Sie den Puzzler Kommilitonen/Freunden empfehlen? (Auswahl: „Ja“/„Nein“)
38. Q38: Warum würden Sie den Puzzler (nicht) weiterempfehlen?

ESSENCE NAVIGATOR

- 39. Q39: Würden Sie ein Werkzeug wie den Essence Navigator in einem Ihrer kommenden Projekte einsetzen wollen? (Auswahl: „Ja“/„Nein“)
- 40. Q40: Sollte ein Werkzeug wie der Essence Navigator standardmäßig in Projekten in Lehrveranstaltungen eingesetzt werden? (Skala 1-5: 1=„gar nicht“, 5=„unbedingt“)
- 41. Q41: Warum würden Sie ein Werkzeug wie den Essence Navigator (nicht) einsetzen wollen?

SEMAT ESSENCE

- 42. Q42: Wie vertraut fühlten Sie sich VOR dem Spiel mit SEMAT Essence? (Skala 1-5: 1=„gar nicht“, 5=„sehr vertraut“)
- 43. Q43: Wie vertraut fühlen Sie sich NACH dem Spiel mit SEMAT Essence? (Skala 1-5: 1=„gar nicht“, 5=„sehr vertraut“)
- 44. Q44: Haben Sie den Eindruck, dass Ihnen die Konzepte aus SEMAT Essence und dem Essence Kernel in Ihren künftigen Projekten helfen können? (Skala 1-5: 1=„gar nicht“, 5=„unbedingt“)
- 45. Q45: Wollen Sie SEMAT Essence in künftigen Projekten einsetzen? (Skala 1-5: 1=„gar nicht“, 5=„unbedingt“)
- 46. Q46: Hätten Sie gern früher etwas über SEMAT Essence gewusst? (Auswahl: „Ja“/„Nein“)

WAS MÖCHTEN SIE UNS NOCH MITTEILEN?

- 47. Q47: Welche weiteren Anmerkungen und/oder Vorschläge zu Essence, dem Puzzler und/oder dem Simulationsspiel haben Sie?

LITERATURE REVIEW—STUDIES OF SIMULATION AND DGBL IN SOFTWARE ENGINEERING EDUCATION AFTER 2013

This chapter provides results of a literature review conducted to be informed about recent developments in the field since existing comprehensive SLRs do provide only studies before 2014.

F.1 DATA SOURCES AND SEARCH STRATEGY

Basically following the meta-studies of Wangenheim and Shull[301] and Jiang et al.[126] a literature review of the years 2014-2016 was conducted. To keep up to date with current developments in the field, the search for new publications was regularly updated.

The author used IEEEExplore, the ACM Digital Library, ScienceDirect, and SpringerLink. Following search strings were used

- In IEEEExplore
 - ((software engineering education) OR (software project management education)) AND (game OR simulation)
- In ACM Digital Library
 - acmdlTitle:(+("software engineering" "project management") +(game simulation)) OR recordAbstract:(+("software engineering" "project management") +(game simulation) +(process method) +education)
- In ScienceDirect
 - ((TITLE-ABSTR-KEY("software engineering") OR TITLE-ABSTR-KEY("software project management")) AND (TITLE-ABSTR-KEY("game") OR TITLE-ABSTR-KEY("simulation")) AND TITLE-ABSTR-KEY("education"))
- In SpringerLink
 - ("software project management" OR "software engineering") AND ("education" OR "training") AND ("game" OR "simulation") AND ("software process" OR "software method")

Widely following the procedure of Wangenheim and Shull[301], English-language articles on games and simulations for software engineering education that were available via the databases mentioned above published after 2013 were considered.

Only papers published in peer-reviewed conference proceedings or journals were included. Included were games, game-like simulations, or contributions to them. Of interest were studies focusing on the area of software process, SE method, or SE management that got utilized for software engineering education.

Excluded were

- general workshop descriptions delivered by their organizers,
- problem/project-based exercises that sometimes get referred to as a simulation.

The goal of this literature review is to present most recent developments in the field. To capture even early developments, the selection criteria are less strict than those of Wangenheim and Shull[301] as well as Jiang et al.[126] and include studies presenting learning games or game-like simulations that were not yet evaluated thoroughly. Observations show that a number of preceding studies were followed by extensive evaluations in subsequent years (cf. 2.12).

Data Source	Retrieved studies	Selected studies
IEEEExplore	82	12
ACM DL	73	4
ScienceDirect	3	1
SpringerLink	68	0

Tabelle F.1: Study Selection By Data Sources

F.2 RESULTS

The results of the literature review are presented in table F.2 each with the name of the study, a short description, the focus of the study, and its chosen evaluation approach.

Tabelle F.2: Identified Simulation and DGBL Studies In Software Engineering Education Focussed on SE Process/Method

Year	Study	Short description	Focused on	Evaluation
2014	A general framework for software project management simulation games[171]	„we propose a general framework and a method that can be used for developing project management simulation games for an educational purpose in the software engineering domain. We developed a general simulation model including aspects of ergonomic analysis, work psychology, learning theory and software engineering. With this, we are able to simulate not only the working processes but also the character of the developers.“[171]	project management simulation games for an educational purpose in the software engineering domain	none reported, done in extending work[172]

continued on next page

Year	Study	Short description	Focused on	Evaluation
2014	GSD Sim: A Global Software Development Game[185]	„We developed a serious game, called "GSD Sim", that allows players to manage a globally distributed software project. Players allocate teams of programmers to different locations around the world, and assign these teams to develop modules that comprise the software product. A simulator generates events, such as integration failures or requirements misunderstandings that cause project delays, players can make tactical and strategic interventions to address and prevent adverse events.“[185]	Global Software Development (GSD)	„At the time of this writing, the global distance model has not been validated;“[185]
2014	InspectorX: A game for software inspection training and learning[216]	„InspectorX, a serious game for learning and training on software inspections, whose design accounts for an optimized cognitive load by offering different levels of difficulty.“[216]	software inspection activities	experiment with 39 undergraduate students, pre-/post-exams, follow up questionnaire (n=22)
2014	Evaluating software engineering simulation games: The UGALCO framework[204]	based on SPIAL[201] game, „We applied this framework for the evaluation of a specific simulation game. Results indicate that this framework can be used to gain better and more understanding of simulation games aspects.“[204]	evaluation of SPIAL[201]	using the proposed framework to evaluate SPIAL[201] (evaluated before through questionnaire and experiment)
2014	Board Game as a Tool to Teach Software Engineering Concept—Technical Debt[89]	„impart the concept of technical debt in a real world setting to students through the board game 'Hard Choices'.“	SE concept of technical debt	experiment, pre-/post-test

continued on next page

Year	Study	Short description	Focused on	Evaluation
2015	Project management game2D (PMG-2D): A serious game to assist software project managers training [153]	„an educational serious game that aims to assist inexperienced software project managers to be trained, considering cost, time, risk and human resources management areas. The PMG-2D simulates a real software development environment where the player, acting as a project manager, goes through all basic phases of a software project lifecycle. There are many roles in the team that have to be managed, and the members have different personalities in order to challenge the player when dealing with people and their conflicts and performance, among other characteristics.“[153]	software project management, different characteristics of project members	questionnaire with participants of a SE course (n=25)
2015	Experimental Evaluation of a Serious Game for Teaching Software Process Modeling[52]	„In the DesigMPS game, the student models a software process from an SPI perspective, based on the Brazilian SPI model (MPS.BR).“[52]	Software Process Modeling (SPM)	Experiment, pre-/post-test
2015	Using simulation games to teach global software engineering courses[307]	utilizing SimSE by implementing „a model for distributed global software development simulation games. The model includes factors like time zones, cultural diversity of users (mainly Hofstede’s culture dimensions are used), location barriers and gender issues.“[307]	Global Software Engineering	none reported, „Control groups from different cultures and backgrounds <i>will be used</i> to test the game.“[307]
2015	Discovering the essence of Software Engineering an integrated game-based approach based on the SEMAT Essence specification[214]	early concept description, „Key objectives of this approach are to sensitize students for the diversity of dimensions that have to be taken into account in a SE endeavor, to provide a valuable guidance for using SE methods inside and outside of their curriculum and to enable students to transfer their newly acquired knowledge to other contexts.“[214]	SEMAT Essence	none reported since it is an early description of concept

continued on next page

Year	Study	Short description	Focused on	Evaluation
2015	Process simulation for software engineering education[126]	„The objective of this research is to present the latest state-of-the-art of this area, and more importantly provide practical support for the effective adoption of SPS in educational context. We conducted an extended Systematic Literature Review (SLR) based on our previous reviews. The review identified 42 primary studies from 1992 to 2013.“[126]	comprehensive Systematic Literature Review (SLR)	
2016	Modeling human behavior for software engineering simulation games[173]	„We present a new decision-making model based on findings of psychology, which can be used for simulating a more realistic human behavior. We use heuristics for calculating the motivational force of all potential actions an employee has, in order to decide which he will choose. This calculation is not only based on the project's state and schedule, but also on emotional factors like the preferences and aversions of the employee.“[173]	project management simulation games for an educational purpose in the software engineering domain	none reported, „plan an empirical evaluation with students to verify the use of our approach for educational purposes.“[173]
2016	GSDgame: A Serious Game for the Acquisition of the Competencies Needed in GSD[293]	„presents a serious game called "GSDgame" with which some of the competencies needed in GSD can be acquired. The game simulates scenarios that usually occur in the overall development of a software project, thus enabling the user to become aware of the problems concerning GSD and gain some experience in solving these problems.“[293]	Global Software Development (GSD)	„evaluation was performed by an expert in serious games“[293]
2016	Strengthening study skills by using ERPsim as a new tool within the Pupils' academy of serious gaming[291]	utilizing ERPsim, a business planning simulation game based on an ERP system (SAP) to promote study skills, primarily time management and teamwork of future students, "the pupils learn how to deal with the SAP platform and train skills like decision making, analysis, strategy development, data processing and presentation"[291]	Enterprise Resource Planning (ERP)	self-assessment of participants in questionnaire
2016	The pupils' academy of serious gaming: Strengthening study skills with ERPsim[292]	utilizing ERPsim, an online SAP ERP Simulation Game, „examinations include the activities related to the three study skills methods of science, self-competences, and a clear idea of studies at university“ [292]	Enterprise Resource Planning (ERP)	self-assessment of participants in questionnaire

continued on next page

Year	Study	Short description	Focused on	Evaluation
2016	Software Engineering Management Education through Game Design Patterns[86]	utilizing SimSE (cf. section 2.5.1) for study, „presents a process to identify the game design patterns that can be effective for teaching software engineering, specifically the software project management topic.“[86]	game design patterns	Validation of the learning and teaching functions through surveys to SE education specialists, empirical study with computer science students to validate the identified game design patterns
2016	Smart decisions: an architectural design game[51]	„a game that aids in teaching architecture design, specifically design employing the Attribute-Driven Design method.“[51]	software architecture	several pilot game sessions, feedback questionnaire
2016	Teaching university students Kanban with a collaborative board game[108]	„GetKanban v4.0 is a collaborative, physical board game that is aimed to teach the basics of Kanban.“[108]	Kanban	2 subsequent classes, questionnaire and learning diaries (n=57)[108]

Tabelle F.2: Identified Simulation and DGBL Studies In Software Engineering Education Focussed on SE Process/Method

SELBSTÄNDIGKEITSERKLÄRUNG

Ich erkläre, dass ich die eingereichte Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe..

Rostock, 2. November 2016

Jöran Pieper



RÉSUMÉ

PERSONAL DATA

Name	Jöran Pieper
Date of Birth	11.04.1975
Place of Birth	Greifswald, Germany
Nationality	German

EDUCATION AND PROFESSIONAL EXPERIENCE

since 10.2013	Ph.D. student at the Faculty of Computer Science and Electrical Engineering, University of Rostock, Germany
since 09.2010	Lecturer and Research Associate at School of Electrical Engineering and Informatics, University of Applied Sciences Stralsund, Stralsund, Germany
03.2006 - 09.2010	Laboratory Engineer at School of Electrical Engineering and Informatics, University of Applied Sciences Stralsund, Stralsund, Germany
09.2002 - 03.2006	IT Specialist, Deutsche Post ITSolutions GmbH, several locations, Germany
12.2002	Diplom-Wirtschaftsinformatiker (FH) (Business Informatics graduate)
09.1998 - 12.2002	Study of Business Informatics at the School of Business Administration, University of Applied Sciences Stralsund, Stralsund, Germany
06.1996 - 08.1997	Employee of the Business Administration Department, Controlling Department, Sparkasse Vorpommern, Greifswald, Germany
09.1993 - 08.1996	Apprenticeship as bank employee, Sparkasse Vorpommern, Greifswald, Germany
08.1993	Abitur (high-school diploma), Alexander-von-Humboldt-Gymnasium, Greifswald, Germany

THESIS STATEMENTS

1. SEMAT Essence facilitates the achievement of learning objectives demanded by curriculum guidelines.
2. Software Engineering (SE) students, who got familiar with SEMAT Essence, esp. the Essence kernel, want to utilize it in future endeavors.
3. An educational interactive simulation model as well as a intrinsic digital learning game providing an hazard-free environment to understand and apply factual, conceptual, procedural, and meta-cognitive knowledge can be built on top of SEMAT Essence.
4. An approach integrating DGBL activities deeply into a SE course and curriculum fosters students' competencies demanded by curriculum guidelines.
5. The cognitive load of students in their course project work can be lowered by preceding DGBL activities.
6. SE students can be provided with concepts and tools to support keeping a holistic view on their course/capstone project right from its start.
7. SE students, who experienced supporting orientation and guidance provided by concepts and tools in a course project, appreciate that support and develop an attitude wanting to utilize those concepts and tools in future projects.
8. Constructivist approaches, esp. based on social constructivism, are not utilized to their full potential in existing DGBL approaches in SE education.
9. Social constructivist instructional approaches contribute to learning success in DGBL approaches.
10. DGBL approaches are qualified to trigger and foster social interaction—online as well as offline.
11. An intrinsic learning game tightly coupled to the underlying domain does not need mandatory “levels and badges” and may draw on the underlying domain to provide rewards to players.
12. A digital learning game can provide both motivating competition *and* collaboration fostering reflection.

13. SE students, who experienced supporting orientation and guidance provided by concepts and tools in an intrinsic simulation game tightly coupled to the underlying domain, appreciate that support and develop an attitude wanting to utilize those concepts and tools in following real project work.
14. Scaffolding by integrating learning material as well as real-world tools into digital learning games facilitates the acquisition/construction of new knowledge in DGBL approaches of SE education.
15. The DEVS simulation formalism provides a thorough theoretical foundation as well as the needed flexibility, provided by the capability to compose hierarchical models, to build interactive educational simulation models based on SEMAT Essence to drive a digital learning game.
16. The DEVS simulation formalism can be reasonably combined with Finite State Machines to represent complex inherent logic of atomic and coupled DEVS models.